```
int k ;
int address ;
int *indxs = 0 ;
int skipIt ;
int numHits = 0 ;
unsigned char **bitArray ;
int size ;
int what ;
int this ;
        Init();
        if ( !( bitArray = (unsigned char
**)UTL_MEM_CALLOC(bitset->numVariationSites,

sizeof(unsigned char *))) )
                goto AddTraceback ;
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
        {
/*
** We only want to count the fragments for the site that is being exploded.
*/
                if ( ( site != -1 ) && ( i != site )  )
                        continue ;
                size = ( bitset->actuallSizes[i] + 7 ) / 8 ;
                if ( !(bitArray[i] = (unsigned char
*)UTL_MEM_CALLOC(size,sizeof(unsigned char))))
                        goto AddTraceback ;
        }
        for ( address = -1 , i = 0 ; i < bitset->totalSelected  ; i++ )
        {
                address = IHBFindNextOne(bitset->bitset,address+1);
                BitSetAddressToIndexes(bitset,address,&indxs,0);
/*
** The sites that have already been expanded will constraint what hits
** we find.
*/
        if ( numFixedSites )
        {
            skipIt = 0 ;
            for ( k = 0 ; k < bitset->numVariationSites ; k++ )
            {
                if ( fixedSitesIndexes[k] == -1 )
                    continue ;
/*
** our hit index matches our constraint index.
*/
                if ( fixedSitesIndexes[k] != indxs[k] )
                {
                    skipIt = 1  ;
                    break;
                }
            }
            if ( skipIt )
                continue ;
        }
        numHits++ ;
        for ( j = 0 ; j < bitset->numVariationSites  ; j++ )
        {
            if ( ( site != -1 ) && ( j != site )  )
                continue ;
            what = indxs[j] % 8;
            this = indxs[j] / 8;
            bitArray[j][this] |= setbits[what];
        }
    }
#if 0
/*
```

```
** Figure out how many hits there are for this site.
*/
    for ( k = 0 ; k < bitset->numVariationSites  ; k++ )
    {
        if (  site != k )
            continue ;
        size = (-bitset->actuallSizes[k] + 7 ) / 8 ;
        numFragmentsPerSite[k] = 0 ;
        for ( j = 0 ; j < size ; j++ )
            numFragmentsPerSite[k] += nbits[bitArray[k][j] & 255];
    }
#endif
/*
** Now get the indexes for all the hits.
*/
    if ( !( (*hitIndexes) = (int *)UTL_MEM_CALLOC(numHits,
                                                              sizeof(int)))
)
        goto AddTraceback ;
    numHits = 0 ;
    for ( k = 0 ; k < bitset->numVariationSites  ; k++ )
    {
    if ( site == -1 )
        {
            if ( fixedSitesIndexes[k] != -1 )
                continue ;
        }
        else
        {
            if ( site != k )
                continue ;
        }
        siz  = ( bitset->actuallSizes[k] + 7 ) / 8 ;
        for ( i = 0 ; i < size ; i++ )
        {
            if ( bitArray[k][i] )
            {
/*
** If any bit is set in the byte then we need to figure out what the hits are.
*/
                for ( j = 0 ; j < 8  ; j++ )
                {
                    if ( bitArray[k][i] & setbits[j] )
                    {
                        (*hitIndexes)[numHits++] = i * 8 + j ;
                    }
                }
            }
        }
    }
    for ( i = 0 ; i < bitset->numVariationSites  ; i++ )
        if ( bitArray[i] )
            UTL_MEM_FREE(bitArray[i]);
    UTL_MEM_FREE(bitArray );
    return numHits  ;

AddTraceback :
    return 0 ;
}


static int GetPartialProductsStats( struct BitSetFileStruct *bitset , int numFixedSites, int
*fixedSitesIndexes, int *numProducts, int *numFragmentsPerSite)
{
int i ;
int j ;
int k ;
int address ;
int *indxs = 0 ;
int skipIt ;
int numHits = 0 ;
unsigned char **bitArray ;
int size ;
int what ;
int this ;
        Init();
        if ( !( bitArray = (unsigned char
**)UTL_MEM_CALLOC(bitset->numVariationSites,

sizeof(unsigned char *))) )
                goto AddTraceback ;
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
        {
/*
** We only want to count the fragments for the sites that are not being
** exploded.
*/
                if ( fixedSitesIndexes[i] != -1 )
                        continue ;
                size = ( bitset->actualSizes[i] + 7 ) / 8 ;
                if ( !(bitArray[i] = (unsigned char
*)UTL_MEM_CALLOC(size,sizeof(unsigned char))))
                        goto AddTraceback ;
        }
        for ( address = -1 , i = 0 ; i < bitset->totalSelected  ; i++ )
        {
                address = IHBFindNextOne(bitset->bitset,address+1);
                BitSetAddressToIndexes(bitset,address,&indxs,0);
/*
** The sites that have already been expanded will constraint what hits
** we find.
*/
        if ( numFixedSites )
        {
            skipIt = 0 ;
            for ( k = 0 ; k < bitset->numVariationSites ; k++ )
            {
                if ( fixedSitesIndexes[k] == -1 )
                    continue ;
/*
** our hit index matches our constraint index.
*/
                if ( fixedSitesIndexes[k] != indxs[k] )
                {
                    skipIt = 1  ;
                    break;
                }
            }
            if ( skipIt )
                continue ;
        }
        numHits++ ;
        for ( j = 0 ; j < bitset->numVariationSites  ; j++ )
        {
            if ( fixedSitesIndexes[j] != -1 )
                continue ;
            what = indxs[j] % 8;
            this = indxs[j] / 8;
            bitArray[j][this] |= setbits[what];
        }
    }
    for ( k = 0 ; k < bitset->numVariationSites  ; k++ )
    {
        if ( fixedSitesIndexes[k] != -1 )
            continue ;
        size = ( bitset->actualISizes[k] + 7 ) / 8 ;
        numFragmentsPerSite[k] = 0 ;
        for ( j = 0 ; j < size ; j++ )
            numFragmentsPerSite[k] += nbits[bitArray[k][j] & 255];


    }
    *numProducts = numHits ;
    for ( i = 0 ; i < bitset->numVariationSites  ; i++ )
        if ( bitArray[i] )
            UTL_MEM_FREE(bitArray[i]);
    UTL_MEM_FREE(bitArray );
    return numHits  ;

AddTraceback :
    return 0 ;
}

static int GetPartialProducts( struct BitSetFileStruct *bitset , int numFixedSites, int
*fixedSitesIndexes, int whichSite, int **siteIndexes)
{
int i ;
int j ;
int k ;
int address ;
int *indxs = 0 ;
int skipIt ;
int numHits = 0 ;
    Init();
    for ( address = -1 , i = 0 ; i < bitset->totalSelected  ; i++ )
    {
        address = IHBFindNextOne(bitset->bitset,address+1);
        BitSetAddressToIndexes(bitset,address,&indxs,0);
/*
** The sites that have already been expanded will constraint what hits
** we find.
*/
        if ( numFixedSites )
        {
            skipIt = 0 ;
            for ( k = 0 ; k < bitset->numVariationSites ; k++ )
            {
                if ( fixedSitesIndexes[k] == -1 )
                    continue ;
/*
** our hit index matches our constraint index.
*/
                if ( fixedSitesIndexes[k] != indxs[k] )
                {
                    skipIt = 1  ;
                    break;
                }
            }
            if ( skipIt )
                continue ;
        }
        numHits++ ;
fprintf(stderr,"Got a hit on %d %d %d\n",address,indxs[0],indxs[1]);
    }
    return numHits  ;

AddTraceback :
    return 0 ;
```

```
}

static GetFragmentsUsedInASite( struct BitSetFileStruct *bitset , int whichSite , int **
indxs)
{
unsigned char *bitArray ;
int i ;
int j ;
int size ;
int *address ;
int numHits = 0 ;
int bit ;
        if (!(address = (int
*)UTL_MEM_CALLOC(bitset->numFragsInEachSite[whichSite],
                                                                sizeof(int))) )
                goto AddTraceback ;
/*
** Figure out how many ints there are in this bitset.
*/
        size = ( bitset->actuallSizes[whichSite] + 7 ) / 8 ;
        for ( bitArray = bitset->fragmentBitset[whichSite], i = 0 ; i < size ; i++ )
        {
                if ( bitArray[i] )
                {
/*
** If any bit is set in the byte then we need to figure out what the hits are.
*/
                        for ( j = 0 ; j < 8  ; j++ )
                        {
                                if ( bitArray[i] & setbits[j] )
                                {
                                        address[numHits++] = i * 8 + j ;
                                }
```

```
                    }
                }
            }
            (*indxs) = address ;
            return numHits ;
    AddTraceback :
            return 0 ;
    }


    static struct BitSetFileStruct *ReadAndAllocate(char *fileName ,int offset )
    {
    struct BitSetFileStruct *bitset ;
            if ( !(bitset = (struct BitSetFileStruct *)UTL_MEM_CALLOC(1,
                                    sizeof(struct BitSetFileStruct ))))
                    goto AddTraceback ;
            if ( !ReadCheckPointFile(fileName,
                                                    offset,

    &(bitset->masterFileInfo.masterFilePathName),

    &(bitset->masterFileInfo.masterRecNo),
                                                    &(bitset->programInfo.programName),
                                                    &(bitset->bitset),
                                                    &(bitset->numVariationSites),
                                                    &(bitset->actualISizes),
                                                    &(bitset->allocSizes),
                                                    &(bitset->totalSelected),
                                                    &(bitset->numFragsInEachSite),
                                                    &(bitset->masterFileInfo),
                                                    &(bitset->programInfo.bufferSize),
                                                    NULL))
                    goto AddTraceback ;
```

```
        return bitset ;
AddTraceback :
        return ( struct BitSetFileStruct *)NULL ;
}

static int ReadBitsetCoreInfo(void *bs, char **masterFileName, int *masterRecno, char
**core, char **xrString, int *numSites, char ***xFileNames )
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *) bs;
int    recNo ;
FILE   *fp ;
int    i ;
int         found = 0 ;
char    *line ;
char    *cp ;
char    *cp1 ;
        *numSites = bitset- > numVariationSites ;
        *masterFileName = bitset- > masterFileInfo.masterFilePathName;
        *masterRecno     = bitset- > masterFileInfo.masterRecNo;
        if ( !((*xFileNames) = (char **)UTL_MEM_CALLOC(*numSites,

sizeof(char *)) ))
                goto AddTraceback ;
        for ( i = 0 ; i < *numSites ; i++ )
                (*xFileNames)[i] =
UTL_STR_SAVE(bitset- > masterFileInfo.x_FileName[i]);
/*
** Open the core file and read in the core and parse out the XRstring.
*/
        if ( !(fp = fopen(bitset- > masterFileInfo.corefilePathName,"r")) )
                goto UnableToReadCore ;
        recNo = 0 ;
        found = 0 ;
        while ( -1 != UTL_SCAN_GETS( fp, "\\", "#", &line))
        {
                recNo++ ;
                if ( recNo ==        bitset->masterFileInfo.startCore )
                {
                        found = 1 ;
                        break;
                }
        }
        if (!found )
                goto UnableToReadCore ;
/*
** Replace all occurances of Y_0x with Xx.
*/
        (*core ) = UTL_STR_SAVE(line);
        cp = strstr(line,"XRLIST=");
        if ( !cp )
                (*xrString) = UTL_STR_SAVE("");
        else
        {
/*
** Skip the first double quote.
*/
                cp += 8 ;
/*
** Go find the end of double quotes.
*/
                cp1 = cp ;
                while ( (*cp)  != '"')
                        cp++ ;
                *cp = 0 ;
                (*xrString) = UTL_STR_SAVE(cp1);
        }
        fclose(fp);
        return 1 ;
    UnableToReadCore :
        fprintf(stderr,"ReadBitsetCoreInfo() -- Unable to read core %s %d\n",
        bitset->masterFileInfo.corefilePathName,
        bitset->masterFileInfo.startCore);
    AddTraceback :
        fprintf(stderr,"ReadBitsetCoreInfo() -- Unable to read core info\n");
        return 0 ;
    }

    static int ReadMasterCoreInfo(char *masterFile, int index, char **core, char **xrString,
    int *numSites, char ***xFileNames )
    {
    int    recNo ;
    FILE   *fp ;
    int    i ;
    int        found = 0 ;
    char    *line ;
    char    *cp ;
    char    *cp1 ;
    char *prefix = (char *)NULL ;
    char *coreFile = (char *)NULL ;
    char *fpFileName = (char *)NULL ;
    int  fpOffset ;
    int  mBits ;
    int  lBits ;
    int  startCore ;
        *numSites = 2 ; /* fixed for now */
        if ( !((*xFileNames) = (char **)UTL_MEM_CALLOC(*numSites,

    sizeof(char *)) ))
```

```
            goto AddTraceback ;
/*
** Get the master file info.
*/
        if ( !GetMasterRecordHeader(masterFile,
                                                index,
                                                &prefix,
                                                &mBits,
                                                &lBits,
                                                &coreFile,
                                                &startCore,
                                                &(*xFileNames)[0],
                                                &(*xFileNames)[1],
                                                numSites,
                                                &fpFileName,
                                                &fpOffset))
            goto AddTraceback ;
/*
** Open the core file and read in the core and parse out the XRstring.
*/
        if ( !(fp = fopen(coreFile,"r")) )
            goto UnableToReadCore ;
        recNo = 0 ;
        found = 0 ;
        while ( -1 != UTL_SCAN_GETS( fp, "\\", "#", &line))
        {
            recNo++ ;
            if ( recNo ==        startCore )
            {
                found = 1 ;
                break;
            }
        }
        if (!found )
                goto UnableToReadCore ;
        (*core ) = UTL_STR_SAVE(line);
        cp = strstr(line,"XRLIST=");
        if ( !cp )
                (*xrString) = UTL_STR_SAVE("");
        else
        {
/*
** Skip the first double quote.
*/
                cp += 8 ;
/*
** Go find the end of double quotes.
*/
                cp1 = cp ;
                while ( (*cp)  != '"')
                        cp++ ;
                *cp = 0 ;
                (*xrString) = UTL_STR_SAVE(cp1);
        }

        fclose(fp);
    if ( coreFile )
                UTL_MEM_FREE(coreFile);
    if ( fpFileName )
                UTL_MEM_FREE(fpFileName);
    if ( prefix )
                UTL_MEM_FREE(prefix);
        return 1 ;
UnableToReadCore :
        fprintf(stderr,"ReadMastersetCoreInfo() -- Unable to read core %s %d\n",
                coreFile,startCore);
```

```
AddTraceback :
        fprintf(stderr,"ReadMastersetCoreInfo() -- Unable to read core info\n");
    if ( coreFile )
                UTL_MEM_FREE(coreFile);
    if ( fpFileName )
                UTL_MEM_FREE(fpFileName);
    if ( prefix )
                UTL_MEM_FREE(prefix);
        return 0 ;
}

static void DeallocateBitset( struct BitSetFileStruct *bitset )
{
int i ;
        if ( bitset- > masterFileInfo.masterFilePathName )
                UTL_MEM_FREE(bitset- > masterFileInfo.masterFilePathName);
        if ( bitset- > masterFileInfo.corefilePathName )
                UTL_MEM_FREE(bitset- > masterFileInfo.corefilePathName);
        if ( bitset- > masterFileInfo.fingerFileName )
                UTL_MEM_FREE(bitset- > masterFileInfo.fingerFileName);
        if ( bitset- > masterFileInfo.prefixForFiles )
                UTL_MEM_FREE(bitset- > masterFileInfo.prefixForFiles);
        for ( i = 0 ; i < bitset->masterFileInfo.numVariationSites ; i++ )
                UTL_MEM_FREE(bitset- > masterFileInfo.x_FileName[i]);
        if ( bitset- > masterFileInfo.x_FileName)
                UTL_MEM_FREE(bitset- > masterFileInfo.x_FileName);
        if ( bitset- > programInfo.programName )
                UTL_MEM_FREE(bitset- > programInfo.programName);
        if ( bitset- > programInfo.buffer )
                UTL_MEM_FREE(bitset- > programInfo.buffer);
        IHBDestroy(bitset- > bitset);
        if ( bitset- > actuallSizes )
                UTL_MEM_FREE(bitset- > actuallSizes);
        if ( bitset->allocSizes )
                UTL_MEM_FREE(bitset->allocSizes);
        if ( bitset->numFragsInEachSite )
                UTL_MEM_FREE(bitset->numFragsInEachSite);
        UTL_MEM_FREE(bitset);
        bitset = (struct BitSetFileStruct *) NULL ;
}_
void CS_PRDCT_BITSET_DUMP( struct BitSetFileStruct *bitset )
{
int i ;
int indx ;
int indx1 ;
int indx2 ;
        fprintf(stderr,"Master file name :
%s\n",bitset->masterFileInfo.masterFilePathName);
        fprintf(stderr,"Master file rec  : %d\n",bitset->masterFileInfo.masterRecNo);
        fprintf(stderr,"Program Name     : %s\n",bitset->programInfo.programName);
        fprintf(stderr,"Number of Sites  : %d\n",bitset->numVariationSites);
        fprintf(stderr,"Number Selected  : %d\n",bitset->totalSelected);
        fprintf(stderr,"Actual Sizes     : ");
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
                fprintf(stderr,"%d ",bitset->actuallSizes[i]);
        fprintf(stderr,"\n");
        fprintf(stderr,"Alloc  Sizes     : ");
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
                fprintf(stderr,"%d ",bitset->allocSizes[i]);
        fprintf(stderr,"\n");
        fprintf(stderr,"Num Frags in X?  : ");
/*
** If the number of fragments is zero then we will write -1 to tell others
** to calculate this themselves.
*/
        for ( i = 0 ; i < bitset->numVariationSites ; i++ )
```

```
                fprintf(stderr,"%d ",(bitset->numFragsInEachSite[i] == 0 )?-1:
                                                   bitset->numFragsInEachSite[i]);
        fprintf(stderr,"\n");
        fprintf(stderr,"Selections       : \n");
        indx = -1 ;
        do
        {
                indx = IHBFindNextOne(bitset->bitset,indx+1);
                if ( indx == -1 )
                        break;
                indx1 = indx / bitset->allocSizes[1] ;
                indx2 = indx % bitset->allocSizes[1] ;
                fprintf(stderr,"%d %d\n",indx1 + 1 ,indx2 + 1 );
        } while ( 1 );
}
void CS_PRDCT_BITSET_GET_HITS( struct BitSetFileStruct *bitset , int **indexes)
{
int i ;
int indx ;
int indx1 ;
int indx2 ;
int hitNo = 0 ;
        indx = -1 ;
        do
        {
                indx = IHBFindNextOne(bitset->bitset,indx+1);
                if ( indx == -1 )
                        break;
                indx1 = indx / bitset->allocSizes[1] ;
                indx2 = indx % bitset->allocSizes[1] ;
                indexes[0][hitNo] = indx1 + 1 ;
                indexes[1][hitNo] = indx2 + 1 ;
                hitNo++ ;
        } while ( 1 );
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_OPEN()
**
** Purpose       : Function will read in the header for a CS product bitset.
**
** Usage :
**
** Returns       : A handle to the product bitset info structure or NULL on
**                 error.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                  Date         Description
** ===================     ========
===============
** Fred Soltanshahi        07/26/96     Original version.
**
**-E:
*/
void *CS_PRDCT_BITSET_OPEN( char *bitsetFileName , int offset )
{
struct BitSetFileStruct *bitset ;
        if ( !(bitset = ReadAndAllocate(bitsetFileName,offset)) )
                return (void *)NULL ;
        bitset->totalSelected = IHBCountOnes(bitset->bitset,
                                   0, IHBBitSize(bitset->bitset));
/*
** If the program did not keep track of and output this to the file then we
** need to calculate it ourselves.
*/
    if (  ( bitset->numFragsInEachSite[0] == 0 ) || ( bitset->numFragsInEachSite[0]
== -1 ) )
    {
        CalculateFragsInSties(bitset);
    }
    return (void *)bitset ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_CLOSE()
**
** Purpose       : Function will close a bitset file and cleanup allocated.
**                 memory.
**
** Usage :
**
** Returns       : None.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                Date        Description
** ====================        ========
================
** Fred Soltanshahi        07/26/96      Original version.
**
**-E:
*/
void CS_PRDCT_BITSET_CLOSE( struct BitSetFileStruct *bitset )
{
        DeallocateBitset(bitset);
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_WRITE()
**
** Purpose       : Function will write a bitset into the given file.
**
** Usage :
**
** Returns       : 1 on success or 0 on failure.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                    Date          Description
** ===================       ========
===============
** Fred Soltanshahi        08/02/96      Original version.
**
**-E:
*/
int CS_PRDCT_BITSET_WRITE(char *fileName,char *programName,struct
BitSetFileStruct *productBitset,int progBufferSize,int *progBuffer)
{
        if ( !WriteOutCompressedBSFile(fileName,

productBitset->masterFileInfo.masterFilePathName,
                                                productBitset->masterFileInfo.masterRecNo,
                                                programName,
                                                productBitset->bitset,
                                                productBitset->numVariationSites,
                                                productBitset->actuallSizes,
                                                productBitset->allocSizes,
                                                productBitset->totalSelected,
                                                productBitset->numFragsInEachSite,
                                                progBufferSize,
                                                progBuffer))
                goto AddTraceback ;
        return 1 ;
AddTraceback :
        fprintf(stderr,"CS_PRDCT_BITSET_WRITE()--Unable to write bitset file\n");
        return 0 ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_CREATE()
**
** Purpose       : Function will create an in-memory product bitset from a
**                 master file.
**
** Usage :
**
```

```
** Returns        : A handle to the product bitset info structure  r NULL on
**                  error.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                  Date        Description
** ===================        ========
===============
** Fred Soltanshahi        08/02/96     Original version.
**
**-E:
*/
void  *CS_PRDCT_BITSET_CREATE(char *masterFileName,
                                      int masterRecNumber,
                                      int *initRawBitset)
{
struct BitSetFileStruct *bitset ;
        if ( !(bitset = ReadAndAllocateMaster(masterFileName,

masterRecNumber,
                                                        initRawBitset)) )
                return (void *)NULL ;
        else
                return (void *)bitset ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_SETBITS()
```

```
**
** Purpose        : Function will copy a raw bitset into the ChemSpace product
**                  bitset format.
**
** Usage :
**
** Returns      : 1 on success or zero on failure.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                    Date          Description
** ====================        ========
===============
** Fred Soltanshahi          08/02/96      Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_SETBITS(void *bs, int *rawBS, int numProducts)
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
void   *compressed ;
static int    firstTime = 1 ;
int    i ;
int    total;
char   *cp = (char *)rawBS ;
int rowLength ;
int index1 ;
int index2 ;
int byte ;
int bit ;
int totalSelected = 0 ;
```

```
        if ( firstTime )
        {
                Init();
                firstTime = 0 ;
        }
/*
** Just create a new one.
*/
        if ( bitset->bitset )
                IHBDestroy(bitset->bitset ) ;
        if ( !(bitset->bitset  = CreateCompressedBitSet(rawBS,
                                                                                    0,

bitset->numVariationSites,

bitset->actualISizes,

bitset->allocSizes) ) )
                goto UnableToCreateBitSet ;
        total = bitset->actualISizes[0] ;
        for ( i = 1 ; i < bitset->numVariationSites ; i++ )
                total *= bitset->actualISizes[i] ;
        if ( numProducts == -1 )
/*
** Calculate what products are being set.
*/
        {
                numProducts = 0 ;
                rowLength = bitset->actualISizes[1] ;
                for ( i = 0 ; i < total ; i++ )
                {

                        byte = ( i  ) / 8 ;
```

```
                    bit  = ( i  ) % 8 ;
                    if ( cp[byte] & setbits[bit] )
                        numProducts++ ;
                }
        }
        bitset->totalSelected = numProducts ;
        return 1 ;
UnableToCreateBitSet :
        fprintf(stderr,"CS_PRDCT_BITSET_SETBITS-- Unable to set bit\n");
        return 0 ;
}


/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_TO_RAW ()
**
** Purpose       : Function will copy a ChemSpace product bitset to a
**                 raw bitset format.
**
** Usage :     calloc rawBS before call. useAlloc nonzero to use allocated
**             rather than actual dimensions
**
** Returns       : 1 on success or zero on failure.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                    Date          Description
** ===================        ========
===============
** David Patterson          09/09/96     Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_TO_RAW (void *bs, int *rawBS, int useAlloc)
{
 _CS_PRDCT_BITSET_CONCAT_RAW(bs, rawBS, 0, useAlloc);
  return 1;
}

int CS_PRDCT_BITSET_CONCAT_RAW(void *bs, int *rawBS, int offset,
                               int useAlloc)
{
  int *indxs = 0;
  int address, sum, b;
  struct BitSetFileStruct *bitset = (struct BitSetFileStruct *) bs;
   for ( address = -1 , b = 0 ; b < bitset->totalSelected  ; b++ )
   {
              address = IHBFindNextOne(bitset->bitset,address+1);
              BitSetAddressToIndexes(bitset,address,&indxs,0);
              if (useAlloc)
                FlagProduct(rawBS, 0,0, address+offset);
              else  /* must explicitly calculate the address */
                {sum=_CS_PRDCT_BITSET_INDEXES_TO_INDEX( bitset, indxs) ;
                  FlagProduct(rawBS, 0,0, sum+offset);
                }
   }
   UTL_MEM_FREE(indxs);
   return 1;
}

/*
**+E:
```

```
**
**
** Function Name : CS_PRDCT_BITSET_SELECTED ()
**
** Purpose       : Function will return a ChemSpace bitset's totalSelected
**
** Usage :
**
** Returns       : integet count of selected bits in bitset
**
** Algorithms    : None.
**
** Revision History :
**
** Author                 Date       Description
** ===================    ========
===============
** David Patterson        09/24/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_SELECTED (void *bsvoid )
{
struct BitSetFileStruct  *bs = (struct BitSetFileStruct *) bsvoid;
     return bs->totalSelected;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_REVEAL ()
**
```

```
** Purpose      : Function will return a ChemSpace bitset's struct info to
**                external calling program.
**
** Usage :
**
** Returns      : 1 on success or zero on failure.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                  Date        Description
** ====================       ========
===============
** David Patterson          09/10/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_REVEAL (void *bsvoid,
        char **MasterFile_Bitset,
        int  *StartRec_Bitset,
         int *BitsInAbsentia,
         int *BitsInAbsentiaNoCount,
         char **CoreFile,
         int  *StartCore,
         char **FngrFile,
         char ***Xfiles,
         int **nY,
         FILE **FngrFile_File,
         int  *FingerOff,
         char **ScreenFileName,
         int  *BytesPerFingerPrint,
         int  *WordsPerFingerprint,
        int **query,
        int **FingerCore_FP,
        int *FingerCore_Card    )
{
int i, size;
int *fooi;
struct BitSetFileStruct  *bs = (struct BitSetFileStruct *) bsvoid;
if (MasterFile_Bitset)
  *MasterFile_Bitset   = bs->masterFileInfo.masterFilePathName ;
if (StartRec_Bitset)
  *StartRec_Bitset      =bs->masterFileInfo.masterRecNo ;
if (BitsInAbsentia)
  *BitsInAbsentia        = bs->masterFileInfo.numberOfMissingBits;
if (BitsInAbsentiaNoCount)
  *BitsInAbsentiaNoCount = bs->masterFileInfo.lbits;
if (CoreFile)
  *CoreFile              = bs->masterFileInfo.corefilePathName;
if (StartCore)
  *StartCore             = bs->masterFileInfo.startCore;
if (FngrFile)
  *FngrFile              = bs->masterFileInfo.fingerFileName;
if (Xfiles)
  *Xfiles               = bs->masterFileInfo.x_FileName;
if (nY)
  *nY                   = bs->actualISizes;
if ( FngrFile_File)
  {     if (!((*FngrFile_File) = UTL_FILE_FOPEN((*FngrFile),"r"))) return 0;
              if (!UTL_FILE_FREAD(&i,sizeof(int),1,*FngrFile_File)) return 0; /* nbits in
fp */
              *BytesPerFingerPrint = ( i + 7 ) / 8 ;
              *WordsPerFingerprint = (*BytesPerFingerPrint + 3) / 4;
              (*query) = (int      *) UTL_MEM_ALLOC( *BytesPerFingerPrint);
```

```
                if (!UTL_FILE_FREAD(&i,sizeof(int),1,*FngrFile_File)) return 0; /* record
cnt  */
                if (!UTL_FILE_FREAD(&i,sizeof(int),1,*FngrFile_File)) return 0; /* record
size */
                rewind(*FngrFile_File);
                if (!(fooi = (int *) UTL_MEM_ALLOC( i              ))) return 0;
                size = (3+i)/4 ;
                for ( i=0; i<= *FingerOff; i++)
                    if (!UTL_FILE_FREAD( fooi,sizeof(int),size,*FngrFile_File))
                        return 0;
/*  if ( fooi[1] != 2 + nY_01 * nY_02 ) return 0; */
                if ( ScreenFileName )
                {
                    if (!((*ScreenFileName) = UTL_STR_SAVE(fooi+4))) return 0 ;
                }

                if ( FingerCore_FP )
                {
                    *FingerCore_FP = fooi;
                    if (!UTL_FILE_FREAD( FingerCore_Card,sizeof(int),1,
*FngrFile_File))
                        return 0;
                    if (!UTL_FILE_FREAD(*FingerCore_FP  ,
                                                    sizeof(int),
                                                    *WordsPerFingerprint,
                        *FngrFile_File))
                        return 0;
                }
    }
        return 1;
}

/*
```

```
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_INDEXES_TO_INDEX()
**
** Purpose       : Function will return the right bit given a set of indices
**
** Usage :         all indexes are 0 based.
**
** Returns       : index to use in bitset.
**
** Algorithms    : None.
**
** Revision History : extracted from CS_PRDCT_BITSET_SET_PRD_BIT by
**                    David Patterson
**
** Author                    Date        Description
** ===================       ========
===============
** Fred Soltanshahi          08/02/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_INDEXES_TO_INDEX( struct BitSetFileStruct *bitset,
                                       int *indexes)
{
int   i ;
int   j ;
int   rowLength[MAX_VARIATION_SITES] ;
int   indx = 0 ;
      for ( i = 0 ;  i < bitset->numVariationSites ;  i++  )
      {
            rowLength[i] = 1 ;
```

```
                for ( j = i + 1 ; j < bitset->numVariationSites  ; j++ )
                    rowLength[i] *= bitset->actuallSizes[j] ;
        }
        for ( i = 0 ;  i < bitset->numVariationSites ;  i++ )
        {
                indx += indexes[i] * rowLength[i] ;
        }
        return indx ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_ALLOC_SIZE_INDEXES_TO_INDEX()
**
** Purpose       : Function will return the right bit given a set of indices
**                 it uses the allocated sizes in the bitset to get the info.
**
*/
int  CS_PRDCT_BITSET_ALLOC_SIZE_INDEXES_TO_INDEX( struct BitSetFileStruct
*bitset,
                                        int *indexes)
{
int    i ;
int    j ;
int    rowLength[MAX_VARIATION_SITES] ;
int    indx = 0 ;
        for ( i = 0 ;  i < bitset->numVariationSites ;  i++  )
        {
                rowLength[i] = 1 ;
                for ( j = i + 1 ; j < bitset->numVariationSites  ; j++ )
                    rowLength[i] *= bitset->allocSizes[j] ;
        }
        for ( i = 0 ;  i < bitset->numVariationSites ;  i++ )
        {
                indx += indexes[i] * rowLength[i] ;
        }
        return indx ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_SET_PRD_BIT()
**
** Purpose       : Function will set a product bit with the  given indexes.
**
** Usage :
**
** Returns       : none.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                   Date        Description
** ===================      ========
===============
** Fred Soltanshahi         08/02/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_SET_PRD_BIT(void *bs, int *indexes)
{
```

```
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
int    indx = 0 ;
        indx = CS_PRDCT_BITSET_ALLOC_SIZE_INDEXES_TO_INDEX(bitset,
indexes);
        IHBSet(bitset->bitset, indx );
        bitset->totalSelected++ ;
        return 1 ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_GET_RINFO()
**
** Purpose        : Function will return the Reaction/Reagent info from
**                  the bitset file.
**
** Usage :
**
** Returns        : none.
**
** Algorithms     : None.
**
** Revision History :
**
** Author                  Date         Description
** ====================    ========
================
** Fred Soltanshahi        01/03/97     Original version.
**
**-E:
*/
```

```
int  CS_PRDCT_BITSET_GET_RINFO(void *bs, char **reactionInfo,char ***reagentInfo)
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
       *reactionInfo = bitset- > masterFileInfo.prefixForFiles ;
       *reagentInfo  =  bitset- > masterFileInfo.reagentInfo ;
       return 1 ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_GET_STATS()
**
** Purpose        : Function will return the statistics for a bitset file,
**                  these will include numberOfSites, originalSizes,
**                  numberOfProducts and Number of fragments used at each
**                  variation site.
**
** Usage :
**
** Returns        : none.
**
** Algorithms     : None.
**
** Revision History :
**
** Author                    Date        Description
** ====================      ========
===============
** Fred Soltanshahi          08/05/96    Original version.
**
**-E:
```

```
*/
int  CS_PRDCT_BITSET_GET_STATS(void *bs, int *numSites, int *numProducts,
int **sizes, int **numUsed )
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
        *numSites = bitset->numVariationSites ;
        *numProducts = bitset->totalSelected ;
/*
** Allocate buffers, if they have not been.
*/
        if ( !(*sizes) )
        {
                if ( !((*sizes) = (int *)UTL_MEM_CALLOC(*numSites,sizeof(int))))
                        goto UnableToAllocateMemory ;
        }
        if ( !(*numUsed) )
        {
                if ( !((*numUsed) = (int *)UTL_MEM_CALLOC(*numSites,sizeof(int))))
                        goto UnableToAllocateMemory ;
        }
        memcpy(*sizes, bitset->actuallSizes, sizeof(int) * *numSites );
        memcpy(*numUsed, bitset->numFragsInEachSite, sizeof(int) * *numSites );
        return 1 ;
UnableToAllocateMemory :
        fprintf(stderr,"CS_PRDCT_BITSET_GET_STATS() -- Unable to allocate
memory\n");
        return 0 ;
}


/*
**+E:
**
```

```
**
** Function Name : CS_PRDCT_BITSET_CORE_INFO()
**
** Purpose        : Function will get the xfile and core and xrstring info
**                  from the bitset file.
**
** Usage :
**
** Returns        : 1 on success or 0 on error.
**
** Algorithms     : None.
**
** Revision History :
**
** Author                  Date        Description
** ====================    ========
===============
** Fred Soltanshahi        08/09/96    Original version.
**
**-E:
*/
int CS_PRDCT_BITSET_CORE_INFO(void *bs, char **masterName, int *masterRecno,
char **core, char **xrString, int *numSites, char ***xFileNames )
{
        return ( ReadBitsetCoreInfo(bs,masterName,masterRecno,
                        core,xrString,numSites,xFileNames));
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_PROG_NAME()
```

```
**
** Purpose       : Function will get the program name that produced this bitset.
**
** Usage :
**
** Returns       : 1 on success or 0 on error.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                 Date        Description
** ====================   ========
===============
** Fred Soltanshahi       08/09/96    Original version.
**
**-E:
*/
int CS_PRDCT_BITSET_PROG_NAME(void *bs, char **programName)
{
        *programName =  ((struct BitSetFileStruct *)bs)->programInfo.programName ;
        return 1 ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_MSTR_CORE_INFO()
**
** Purpose       : Function will get the xfile and core and xrstring info
**                 from the master file.
**
```

```
** Usage :
**
** Returns       : 1 on success or 0 on error.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                  Date        Description
** ====================        ========
================
** Fred Soltanshahi        08/09/96    Original version.
**
**-E:
*/
int CS_PRDCT_MSTR_CORE_INFO(char *masterFile, int index, char **core, char
**xrString, int *numSites, char ***xFileNames )
{
        return ( ReadMasterCoreInfo(masterFile,index
,core,xrString,numSites,xFileNames));
}



/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_CREATE_BIT_STRING()
**
** Purpose       : Function will create a compressed version of a raw bit set.
**                 It returns the memory size needed to hold the bitset.
**
** Usage :
```

```
**
** Returns        : pointer to a compressed bitset(this is not a ChemSpace
**                  product bitset but just a compressed bitstring)
**
** Algorithms    : None.
**
** Revision History :
**
** Author                   Date        Description
** ===================        ========
===============
** Fred Soltanshahi          08/06/96     Original version.
**
**-E:
*/
void  *CS_PRDCT_BITSET_CREATE_BIT_STRING( int *rawBits, int offset, int
numVariations, int *sizes, int *allocSizes, int *totalSize)
{
void *compressed ;
       if ( !(compressed = CreateCompressedBitSet(rawBits,
                                                                        offset,

numVariations,
                                                                        sizes,

allocSizes) ) )
              goto UnableToCreateBitSet ;
       *totalSize = IHBRealSize(compressed);
       return compressed ;
UnableToCreateBitSet :
       fprintf(stderr,"CS_PRDCT_BITSET_CREATE_BIT_STRING() -- Unable to create
bitset\n");
       return ( void *)NULL ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_DESTROY_BIT_STRING()
**
** Purpose        : Function will destroy the memory for a bitstring
**                  allocate by the CREATE call above.
**
** Usage :
**
** Returns        : none
**
** Algorithms     : None.
**
** Revision History :
**
** Author                    Date        Description
** ===================      ========
===============
** Fred Soltanshahi         08/06/96    Original version.
**
**-E:
*/
void  CS_PRDCT_BITSET_DESTROY_BIT_STRING( void *bitset)
{
        IHBDestroy(bitset);
}

/*
**+E:
```

```
**
**
** Function Name : CS_PRDCT_BITSET_GETHITS()
**
** Purpose       : Function will return the indexes(into the original X1,X2 files
**                 for the requested number of hits.
**
** Usage :
**
** Returns       : Number of hits found or -1 for error.
**
** Algorithms    : None.
**
** Revision History :
**
** Author                Date        Description
** ===================      ========
===============
** Fred Soltanshahi        08/07/96    Original version.
**
**-E:
*/
int  CS_PRDCT_BITSET_GETHITS( void *bs, int offset, int numberOfHits, int
***hitIndexes)
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
int numFound ;
int numConnections ;
static int *bitAddresses = (int *)NULL ;
static int numBitAddresses = 0 ;
int *indxs = (int *)NULL ;
int start ;
int count ;
int i ;
int j ;
        (*hitIndexes) = (int **)NULL ;
        numConnections = bitset->numVariationSites ;
/*
** Local housekeeping .
*/
        if ( numberOfHits > numBitAddresses )
        {
                if ( !bitAddresses )
                {
                        if ( !(bitAddresses = (int *)UTL_MEM_CALLOC(numberOfHits,

sizeof(int))) )
                                goto UnableToAllocate ;

                }
                else
                {
                        if ( !(bitAddresses = (int *)UTL_MEM_REALLOC(bitAddresses,

numberOfHits* sizeof(int))) )
                                goto UnableToAllocate ;
                }
                numBitAddresses = numberOfHits ;
        }
/*
** Figure out if we have the number of hits he wanted and what their addresses
** are in the bitset file.
**
** We will have to come back and speed this up if it is to slow, but for now
** lets get it working.
**
```

```
*/
/*   start = bitset->firstHitAddress ; */
     start = -1 ; /* start from the begining */
     for ( count = 0 ; count < offset ; count++ )
     {
          start = IHBFindNextOne(bitset->bitset,start+1);
/*
** Lets remember where the first hit is, this should save us some time later.
*/
          if ( bitset->firstHitAddress <= 0 )
               bitset->firstHitAddress = start ;
          if ( start == -1 )
          {
               return 0 ;
          }
     }
/*
** Now lets see how many bits are set from here on.
*/
     for ( numFound = 0 ; numFound < numberOfHits ; numFound++ )
     {
          start= IHBFindNextOne(bitset->bitset,start+1);
          if ( start == -1 )
               break;
          bitAddresses[numFound] = start ;
     }
/*
** Allocate the arrays.
*/
     if ( !(*hitIndexes = (int **)UTL_MEM_CALLOC(numConnections,sizeof(int *))) )
          goto UnableToAllocate ;
     for ( i = 0 ; i < numConnections ; i++ )
     {
```

```
                if ( !((*hitIndexes)[i] = (int *)UTL_MEM_CALLOC(numFound,

sizeof(int ))) )
                        goto UnableToAllocate ;
        }
/*
** Now translate each one of the bitset addresses to the variation site
** indexes.
*/
        for ( i = 0 ; i < numFound ; i++ )
        {
                BitSetAddressToIndexes(bitset,bitAddresses[i],&indxs,0);
                for ( j = 0 ; j < numConnections ; j++ )
                        (*hitIndexes)[j][i] = indxs[j] + 1 ; /* Translate to 1 based indexes */
        }
        if ( indxs )
                UTL_MEM_FREE( indxs );
        return numFound ;
UnableToAllocate :
AddTraceback :
        if ( indxs )
                UTL_MEM_FREE( indxs );
        return -1 ;
}

/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_GET_PARTIAL_HITS()
**
** Purpose       : Function will return the indexes(into the original X1,X2 files
**                 for the requested number of hits.
```

```
**
** Usage :
**
** Returns        : Number of hits found or -1 for error.
**
** Algorithms     : None.
**
** Revision History :
**
** Author                    Date        Description
** ====================     ========
================
** Fred Soltanshahi          08/07/96     Original version.
**
**-E:
*/
int CS_PRDCT_BITSET_GET_PARTIAL_HITS( void *bs, int *numProducts, int site, int
numFixedSites, int *fixedSitesIndexes, int *numFragmentsPerSite, int **hitIndexes )
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
int    total ;
       (*hitIndexes) = (int *)NULL ;
       GetPartialProductsStats( bitset ,
                                                   numFixedSites,
                                                   fixedSitesIndexes,
                                                   &total,
                                                   numFragmentsPerSite);

       (*numProducts ) = GetPartialProductsAddresses(bitset,
                                                  numFixedSites,
                                                  fixedSitesIndexes,
                                                  site,
                                                  hitIndexes);
```

```
    return 1 ;
}


/*
**+E:
**
**
** Function Name : CS_PRDCT_BITSET_GET_PRDCT_PARTIAL_HITS()
**
** Purpose      : Function will return the indexes(into the original X1,X2 files
**                for the requested number of hits.
**
** Usage :         This works when the csln is actually being exploded.
**
** Returns      : Number of hits found or -1 for error.
**
** Algorithms   : None.
**
** Revision History :
**
** Author                  Date       Description
** ===================         ========
===============
** Fred Soltanshahi        08/07/96     Original version.
**
**-E:
*/
int CS_PRDCT_BITSET_GET_PRDCT_PARTIAL_HITS( void *bs, int *numProducts, int
site, int numFixedSites, int *fixedSitesIndexes, int *numFragmentsPerSite, int **hitIndexes
)
{
struct BitSetFileStruct *bitset = (struct BitSetFileStruct *)bs ;
int    total ;
```

```
        (*hitIndexes) = (int *)NULL ;
        GetPartialProductsStats( bitset ,
                                                    numFixedSites,
                                                    fixedSitesIndexes,
                                                    &total,
                                                    numFragmentsPerSite);

        (*numProducts ) = GetPartialProductsAddresses(bitset,
                                                    numFixedSites,
                                                    fixedSitesIndexes,
                                                    site,
                                                    hitIndexes);
      (*numProducts ) = total ;
        return 1 ;
    }

/* +E
Abstract: For Chemspace bitset file call callback with products choices not selected.
Input:
1. This function takes a BitSetFileStruct returned most likely from:
            CS_PRDCT_BITSET_OPEN(char *filename)
2. A void pointer which is passed to callback function.  This is for
        whatever you want.
3. A pointer to function returning:
                        int (void *udata, int numVariants, int *choices ).
        choices is of size numVariants, the choices are zero based, and
        choices[0] is the choice for markush Y_01, choice[1] for Y_02 etc.
        NOTE 1: numVariants of -1 and a null for choices is passed to signify
            the end of the choices excluded, just in case the function
            want to do some special processing at the end.
        NOTE 2: The return value from the callback function is ignored.
Returns:
        Total number of bits excluded.
        -1 upon error.
** Author                    Date          Description
** ====================        ========
===============
** Rob Jilek                 07/26/96      Original version.
*/
int CS_PRDCT_BITSET_ZERO(struct BitSetFileStruct *bitset, void *udata,
int (*ZeroProducts)(void *udata, int numVariants, int *choices ) )
{
        BIT_TRACKING bt[1];
        if ( bitset->numVariationSites <= 0 )
                return -1;
        bt->numVariations = bitset->numVariationSites;
        bt->bitset = bitset;
        bt->call_udata = udata;
        bt->funcptr = ZeroProducts;
        bt->choices = (int *) UTL_MEM_CALLOC(bt->numVariations, sizeof(int) );
        bt->totalExcluded = 0;
                /* The sequence is as follows:
                        IHBRange has a loop to find zeros/ones.
                                It calls RangeCallback
                                        RangeCallback calls ZeroProducts callback.
                        while ( not end of list )
                                call RangeCallback with start and end Range.
                                        for ( i = startRange; i <= EndRange; i++ ) //
RangeCallback
                                                calculate product array.
                                                call ZeroProducts                    //
ZeroCallback
                */

        IHBRange(bitset->bitset, 0, (void *) bt, RangeCallback );
        UTL_MEM_FREE((char *) bt->choices );
        return bt->totalExcluded;
}
/*+I
Synopsis: Gets called for each range of bits set. It then
converts each bit to a product array and calls callback for each.
*/
static int RangeCallback (void *udata, int startRange, int endRange )
{
        BIT_TRACKING *bt = (BIT_TRACKING *) udata;
        int indx;
        int oor;
        int skip;
        void *call_udata;
        int numVar;
        int *choices;
        void *bitset;
        call_udata = bt->call_udata;
        numVar = bt->numVariations;
        choices = bt->choices;
        bitset = bt->bitset;
        for ( indx = startRange; indx <= endRange; )
        {
                skip = BitSetAddressToIndexes(bitset,indx,&choices,&oor);
                if ( !oor )
                {
                        (*bt->funcptr)(call_udata, numVar, choices );
                        bt->totalExcluded++;
                        indx++;
                }
                else
                {
                        if ( skip > 0 )
                                indx += skip;
```

```
                    else
                            indx++;
                }
        }
        (*bt->funcptr)(call_udata,-1, (int *) 0 );              /* Signify end of zeros. */
        return 0;
}

/* +E
Abstract: For Chemspace bitset file call callback with products choices selected.
Input:
1. This function takes a BitSetFileStruct returned most likely from:
                CS_PRDCT_BITSET_OPEN(char *filename)
2. A void pointer which is passed to callback function.  This is for
        whatever you want.
3. A pointer to function returning:
                                int (void *udata, int numVariants, int *choices ).
        choices is of size numVariants, the choices are zero based, and
        choices[0] is the choice for markush Y_01, choice[1] for Y_02 etc.
        NOTE 1: numVariants of -1 and a null for choices is passed to signify
                the end of the choices excluded, just in case the function
                want to do some special processing at the end.
        NOTE 2: The return value from the callback function is ignored.
Returns:
        Total number of bits included.
        -1 upon error.
See Also: CS_PRDCT_BITSET_ZERO
** Author                  Date        Description
** ====================        ========
================
** Rob Jilek               08/19/96    Original version.
*/
int CS_PRDCT_BITSET_ONE(struct BitSetFileStruct *bitset, void *udata,
int (*OneProducts)(void *udata, int numVariants, int *choices ) )
{
        BIT_TRACKING bt[1];
        if ( bitset->numVariationSites <= 0 )
                return -1;
        bt->numVariations = bitset->numVariationSites;
        bt->bitset = bitset;
        bt->call_udata = udata;
        bt->funcptr = OneProducts;
        bt->choices = (int *) UTL_MEM_CALLOC(bt->numVariations, sizeof(int) );
        bt->totalExcluded = 0;
        IHBRange(bitset->bitset, 1, (void *) bt, RangeCallback );
        UTL_MEM_FREE((char *) bt->choices );
        return bt->totalExcluded;
}

#if 0
main(argc,argv)
int argc ;
char *argv[] ;
{
void *h ;
char *masterFileName =
"/home7/fred/work/ADS/dserv/source/dbcsln_des/TestData/Di_300_400.mf" ;
int   masterRecNumber = 1 ;
int  *bitset ;
int  size = ( 300 * 400 + 7 ) / 8 ;
int  i ;
int  j ;
int  indexes[2] ;
char hold[81];
#if 1
        if ( !(h = CS_PRDCT_BITSET_OPEN(argv[1],0)))
        {
                fprintf(stderr,"Unable to open the bitset file %s\n",argv[1]);
                exit ;
        }
        CS_PRDCT_BITSET_DUMP(h);
        CS_PRDCT_BITSET_CLOSE(h);
#else
        if ( !( h =
CS_PRDCT_BITSET_CREATE(masterFileName,masterRecNumber,NULL) ) )
        {
                fprintf(stderr,"Unable to create bitset for %s\n",masterFileName);
                exit ;
        }
        CS_PRDCT_BITSET_WRITE("Test.bs","MyProg",h,0,NULL);

        indexes[0] = 59 ;
        indexes[1] = 129 ;
    CS_PRDCT_BITSET_SET_PRD_BIT(h,indexes);
        indexes[0] = 159 ;
        indexes[1] = 241 ;
    CS_PRDCT_BITSET_SET_PRD_BIT(h,indexes);
        CS_PRDCT_BITSET_WRITE("Test2.bs","MyProg",h,0,NULL);
        bitset = (int *)UTL_MEM_CALLOC(size,sizeof(int));
        bitset[5] = 49 ;
        bitset[1] = 99 ;
    CS_PRDCT_BITSET_SETBITS(h,bitset,-1);
        CS_PRDCT_BITSET_WRITE("Test1.bs","MyProg",h,0,NULL);
        CS_PRDCT_BITSET_CLOSE(h);
#endif
}
#endif
```

Appendix "S"

```
/*****************************************************************************
*/
/*                    topsim                              */
/*****************************************************************************
*/
/*+C
 *
 * This program determines which csln "products" are similar to an input
 * structure, where similarity occurs if the sum of differences in encoded
 * "CoMFA" fields is less than some threshold.
 *
 * The csln components are referenced in a master file with
 * one multiline record per cSLN. Record format is
 *  Reaction class xxxx          (where "Reaction class" is a literal)
 *  reaction_name
 *  number_of_sv_sites
 *  missing_bits_count
 *  hashed_only_missing_bits_count
 *  core_filename
 *  core_filename_index_of_core
 *  fingerprint_filename
 *  offset_into_fingerprint_file
 *  first_sv_file_X1
 *  secod_sv_file_X2             (etc if more than two sv_sites)
 *
 * NOTE -- ALL subsequent entries in the master file whose Reaction class
 * matches the Reaction class of the record referenced by -index are also
 * processed! ("Matching" implies matching of possible other input symbols
 * to components of the Reaction class line.)
 *
 * The input structure is read as encoded fields from stdin (or
 * a named file if provided), one field per line. There
* must be provided (by a SYBYL SPL script), in order:
*
* "number_of_sv_sites" * "number_of_field_types" fields describing the "core" of the
query
* "number_of_sv_sites" - 1 sextets of relative coordinates of core attachment atoms
* "number_of_sv_sites" * "number_of_field_types" fields describing the "side chains"
*
* Options:
*
*    -master name        - name is the file with master file records
*    -bitset name       - name is a result of an earlier search operation
*                       (use EITHER master or bitset)
*
*    -index number       - which sequential record in master file to begin at
*                        OR offset into bitset in a bitset file
*                        (default = 1)
*
*    -reaction name      - records in master file to be processed must have this
*                        class name
*
*    -details  name      - if provided, records in master file to be processed
*                           must have any one of these tokens following its class name
*
*    -distance tan      - tan is the overall similarity threshold
*                       (default is 90.0)
*
*    -cooweight cwt       - weight of the core attachment coordinates,
*                           relative to fields
*
*    -nocore nocore              - do not consider core topomer differences
*                           By default these are considered (required)
*
*    -allcores allc      - process all cores in the core file
*                           By default  nly one core (index in the master file) is processed
*
*    -maxhits max          - stop when max hits are found (default infinity)
*
*    -input filename       - name of file with queries (default stdin)
*
*    -output filename      - specifies the output file for the hit info
*
*     -#                   This flag forces the display of all
*                          options
*
*

*****************************************************************************
/
#include <stdio.h>
#include <signal.h>
#include <ctype.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <math.h>
#include "parseopt.h"
#include "utl_str.h"
#include "utl_mem.h"
#include "utl_file.h"
#include "utl_math.h"
#include "ct.h"
#include "ct_expr.h"
#include "ct_proto.h"
#define GoodExit 0
#define ErrorExit 1
#define Visual(s) {                fprintf s; }
```

```
static FILE        *OutputFile = 0;
static char        *OutputFileBase;
static char        OutputFileName[200];
static int         nOutFiles = 0;  /* number of output files */
static char        *MasterFile = 0;
static char        *BitsetFile = 0;
void               *bitset;
static int         MasterRecord = 1;
static FILE        *MasterFile_File;
static int         StartCore;
static char        *InputSource = 0;
static FILE        *InputSourceFile;
static char        *ReactionNeeded = 0;
static char        *ScratchDetails = 0;
static int         nDetail = 0;
static char        **ReactionDetails = 0;
static char        *XWeights = 0;
static double      *RWeights = 0;
static double      CoreWeight;
static char        *FieldTypes = 0;
static int         nFType = 0;
static char        **FTypes = 0;
static double      *FWeights = 0;
static char        **FOrder = 0; /* temp, for recording L->R order of data
                                    in side chain SLN */
static char        **FROrder = 0;
static char        *Corefile = 0;
static FILE        *CoreFile_File;
static char        *CoreNow;
static char        **Xfile;
static char        **Xname;
static double      Distance = 90.0;
static double      CoreDistance = 0.0;
static double            DWeight = 1.0;
static double            Dist[16][16];
static double            boundary[16];
static double            CXcoords[6], CXdiffsq[6];
static double            searched = 0.0; /* number searched */
static double            combi = 1.0; /* number of side chain combos */
static int              totnout = 0, nout = 0; /* number of products */
static int                *Good_Products = 0; /* product bit set */
static int                *Dead_Products = 0; /* forbidden product bit set */
static int              nR;     /* number of R positions (usually 2) */
static int              *nX;    /* number of product dimensions */
static int                *Xct;  /* used for indexing over all products */
static int              **Xsize; /* bytes per field */
static unsigned char       ****X = 0;   /* csln field (F x R x nX )*/
static unsigned char       ***Xin;  /* target fields */
static unsigned char       ***Y;    /* csln core fields (F x R ) */
static unsigned char       ***Yin;  /* target core fields */
static double              ***X2Y;  /* distances between X and X' */
static int              nSym,   /* number of symmetries in this core */
                          *CoreSyms, /* flags for all matching core symmetries */
                        **SymList; /* symmetry mappings */
static int              DefaultSym[9] = {0, 1, 2, 3, 4, 5, 6, 7, 8};
static int              ReverseSym[2] = {1, 0};
static int              AppendToOutputFile = 0;
static int              NoMorehitsPlease = 0;
static int              UserAborted;
static int              NoCore = 0;
static int              AllCores = 0;
static int              CoreOK = 0;
static int              CoreIsSame = 0;
static int              SideChainOnly = 0;
static int              SideChainsAreSame = 0;
static int              NotBitOutput = 0;
static char                    comline[2048];
static struct ParseOptions Options[] = {
      {"master",     ParseOptString,       &MasterFile,
            "Prefix for all input files" },
      {"bitset",     ParseOptString,       &BitsetFile,
            "Name is the file with bitset records" },
      {"distance",   ParseOptDouble,       &Distance,
            "Field similarity threshold (default 90.0)" },
      {"cooweight",  ParseOptDouble,       &DWeight,
            "Core coord wt, relative to fields (default )" },
      {"index",      ParseOptInt,          &MasterRecord,
            "Which MasterRecord entry 1-n" },
      {"maxhits",    ParseOptInt,          &NoMorehitsPlease,
            "Maximum number of hits before stopping" },
      {"nocore",     ParseOptInt,          &NoCore,
            "Use -nocore to override inclusion of the core differences" },
      {"allcores",   ParseOptInt,          &AllCores,
            "Use -allcores to search all cores provided" },
      {"input",      ParseOptString,       &InputSource,
            "File from which queries will be read( default stdin). "},
      {"output",     ParseOptString,       &OutputFileBase,
            "File to which hit info will be written. "},
      {"notbits",    ParseOptInt,          &NotBitOutput,
            "Use notbits to output as index ASCII instead of std bitset." },
      {"reaction",   ParseOptString,       &ReactionNeeded,
            "Reaction class for topomer search. "},
      {"details",    ParseOptString,       &ScratchDetails,
            "Details further discriminating the reaction class. "},
      {"sidechain",  ParseOptInt,          &SideChainOnly,
            "Use sidechain to search for similiarity in a single sidechain only. "},
      {"fieldtypes", ParseOptString,       &FieldTypes,
            "Names of all field types (optional prefix =weight), space separated. Does
CTOPS if none provided."},
        {"xweights",    ParseOptString,         &XWeights,
                "Weights of varying sites. Must be nR(+core?) individual weights present (if
any)."},
};
int UBS_OUTPUT_MESSAGE() { return 0; }    /* just for compiling OK */
int UIMS2_WRITE_PHOTO() { return 0; }
int lowercase (s) char *s; {while (*s) { if isupper(*s) *s = tolower(*s); s++;}}
static int ParseArguments( argc, argv )
/*+I
 *
 * This function parses the command line arguments.
 *
 * Returns:  1 on a successful command line parse, 0 otherwise.
 *
 * Warnings:
 *
 * Errors:
 *
 * Author      Date           Description
 * ======      ========       ===========
 * G. B. Smith  02-09-93      Original Version
 *
 */
int     argc;
char    **argv;
{
        int     nargs,
                noptions = sizeof( Options )/sizeof(Options[0]);
        nargs = UTL_PARSE_OPT( argc, argv, noptions, Options );
        if( !nargs ) goto SyntaxError;
        return 1;
SyntaxError:
        fprintf( stderr, "Bad command line argument(s)\n" );
        return 0;
}
static int OpenOutputFile()
/*+I
 *
 * Returns:  1 on sucesss, else 0
 *
 */
{
        char    *msg;
        FILE    *fp;
        OutputFile = stdout;
        if( OutputFileBase)
        {
            MakeOutputFileName();
/*
** We need to create output files under the ownership of the REAL user not the
** EFFECTIVE user.  This only applies if setuid options are activated.
*/
{
struct stat statBuff ;
int    uid ;
int    euid ;
        uid = getuid() ;
        euid = geteuid();
    stat(OutputFileName, &statBuff);
/*
** There are two cases
** (1) the file to output to exists
**    Use the ownership of the current owner of the file or if you cant do that
**    do not do anything.
** (2) The file is being created.
**    use the ownership of the REAL user.
*/
    if ( access(OutputFileName, F_OK) == 0 )
    { /* If the file exist and the real user is the owner of the file */
        if ( statBuff.st_uid == uid )
            seteuid(uid);
    }
    else
    { /* Create the file as the REAL user */
        seteuid(uid);
    }
}
    OutputFile = fopen( OutputFileName, (AppendToOutputFile?"a":"wb"));
    if( !OutputFile ) {
        fprintf(stderr,"Error: Failed to open output file \"%s\"\n",
            OutputFileName );
        goto ErrorReturn;
    }
  }
    return 1;
ErrorReturn:
    return 0;
}
static int WhatsTheDifference()
/* builds distance lookup table and initializes default symmetry data structure */
{
 int i, j;
#define pow2(a) ( (a) * (a) )
/* the assignment of codes is based on the following (from gen_pls.c):
  static fpt cutoff[16] = {9999.,  0.,  2.,  4.,  6.,  8.,  10.,  12.,
                            14.,  16.,  18.,  20.,  22.,  24.,  26.,  30.  };
*/
 boundary[0] = 9999.; /* missing data ought never to occur. */
 boundary[1] = -0.1 ;
 for (i=2;i< 15;i++)
   boundary[i] = 2*i-3;
 boundary[15] = 30.0;  /* this is a steep curve with a cutoff at 30! */
 for (i=0;i<16;i++) for (j=0;j<16;j++)
   Dist[i][j] = pow2( boundary[i] - boundary[j]);
 Distance *= Distance;  /* want to test D^2 directly */
 DWeight *= DWeight;
/* allocate once for all conceivable symmetry reorderings */
 if (!(SymList = (int **) UTL_MEM_ALLOC( sizeof( int *) * nR * (nR - 1) / 2) ))
               return 0;
 if (!(CoreSyms = (int *) UTL_MEM_ALLOC( sizeof( int ) * nR * (nR - 1) / 2) ))
               return 0;
 SymList[ 0 ] = DefaultSym;
 SymList[ 1 ] = ReverseSym;
 return 1;
}
static int ReadAField( hex, index, pXP )
/* converts field from external (ASCii hex) format to internal */
char *hex;
int *index;
unsigned char **pXP;
{
   int words, hold;
   char next2[10], *nxhx;
   words = strlen( hex ) / 2;  /* assuming 8-bit bytes */
   if (! *index ) *index = words;
   if ( words != *index ) {
/* bad field (most likely NULL), continue anyway */
        *pXP = (unsigned char *) NULL;
        return 1;
   }
   if (!(*pXP = (unsigned char *) UTL_MEM_ALLOC(words) )) return 0;
   for (words=0, nxhx = hex; words< *index ; words++) {
```

```
            memcpy(next2, nxhx, 2);
            nxhx += 2;
            sscanf( next2, "%2x", &hold );
            *(*pXP + words) = (unsigned char) hold;
    }
    return 1;
  }
  static int RetrieveInput() {
  /* reads the search pattern fields (generated by SYBYL script) */
   int index, R, F;
   char *line;
   double atof();
   if (!InputSource) InputSourceFile = stdin;
     else if (!(InputSourceFile = fopen( InputSource, "r" ) )) {
            fprintf( stdout, "Could not open -input file %s\n", InputSource );
            return 0;
   }
   if (!(Yin = (unsigned char ***) UTL_MEM_ALLOC( sizeof( unsigned char **) * nFType
  )))
           return 0;
   for (F = 0; F < nFType; F++) {
     if (!(Yin[ F ] = (unsigned char **) UTL_MEM_ALLOC( sizeof( unsigned char *) * nR
  )))
           return 0;
     memset( Yin[F], 0, sizeof( unsigned char *) * nR );
   }
  if (!NoCore) {
  /* field types are paired closest! */
   for (index = 0; index < nR; index++) for (F = 0; F < nFType; F++) {
  /* a Field is on a single line, no parsing needed */
          if (-1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &line))
                 return 0;
          if (!ReadAField( line, Xsize[ F ] + index, Yin[ F ] + index )) return 0;
```

```
}
for (index = 0; index < 6; index++) {
     if (-1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &line))
          return 0;
     CXcoords[ index ] = atof( line );
}
if (!(Xin = (unsigned char ***) UTL_MEM_ALLOC( sizeof( unsigned char **) * nFType
)))
   return 0;
for (F = 0; F < nFType; F++) {
  if (!(Xin[F] = (unsigned char **) UTL_MEM_ALLOC( sizeof( unsigned char *) * nR
)))
   return 0;
  memset( Xin[F], 0, sizeof( unsigned char *) * nR );
}
for (index = 0; index < nR; index++) for (F = 0; F < nFType; F++ ) {
/* a Field is on a single line, no parsing needed */
      if (-1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &line))
           return 0;
      if (!ReadAField( line, Xsize[ F ] + index, Xin[ F ] + index )) return 0;
 }
}
 fclose( InputSourceFile );
 return 1;
}
static int InitCore() {
/* readies core file and its input arrays */
  int R, i, F;
  char *foo;

  if (! (CoreFile_File = fopen(Corefile,"r"))) {
      fprintf( stderr, "%s Core file not found.\n", Corefile );
      return 0;
  }
  i=0;
  while ( i < StartCore )
  {
    if ( -1 == UTL_SCAN_GETS( CoreFile_File, "\\", "#", &foo)) return 0;
    if (AllCores) break;
    i++;
  }
  CoreNow = UTL_STR_SAVE( foo );
/* initialize core data structures */
  if (!(Y = (unsigned char ***) UTL_MEM_ALLOC( sizeof( unsigned char **) *
nFType)) )
                return 0;
  for (F = 0; F < nFType; F++) {
    if (!(Y[F] = (unsigned char **) UTL_MEM_ALLOC( sizeof( unsigned char *) * nR)) )

                return 0;
    for (R = 0; R < nR; R++)
      if (!( *( (Y[F]) + R ) = (unsigned char *) UTL_MEM_ALLOC( sizeof( unsigned
char )
                * (*Xsize[F ]) + R ) )) return 0;
  }
  return 1;
}
int CountLines()
{
 int i;
 char *foo;
/* note that CountLines returns one less than the actual number */
 i=0;
 while ( -1 != UTL_SCAN_GETS( InputSourceFile, "\\", "#", &foo)) i++;
 rewind(InputSourceFile);
 return i;
```

```
}
static int initXarrays ()
{
        int F, i;
        if (!(Xfile = (char **) UTL_MEM_ALLOC( sizeof( char* ) * nR ))) return 0;
        if (!(Xname = (char **) UTL_MEM_ALLOC( sizeof( char* ) * nR ))) return 0;
        if (!(nX = (int*) UTL_MEM_ALLOC( sizeof( int ) * nR ))) return 0;
        if (!(Xct = (int*) UTL_MEM_ALLOC( sizeof( int ) * nR ))) return 0;
        for (i = 0; i < nR; i++) { Xfile[i] = 0; Xname[i] = 0; nX[i] = 0; Xct[i] = 0; }
        if (!(X = (unsigned char ****) UTL_MEM_ALLOC( sizeof( unsigned char ***) *
nFType)) )
                return 0;
        for (F = 0; F < nFType; F++) {
           if (!(X[F] = (unsigned char ***) UTL_MEM_ALLOC( sizeof( unsigned char **)
* nR)) )
                return 0;
           memset( X[F], 0, sizeof( unsigned char **) * nR );
        }
        if (!(Xsize = (int **) UTL_MEM_ALLOC( sizeof( int * ) * nFType ))) return 0;
        for (F = 0; F < nFType; F++) {
           if (!(Xsize[F] = (int *) UTL_MEM_ALLOC( sizeof( int ) * nR ))) return 0;
           for (i = 0; i < nR; i++) *(Xsize[F] + i) = 0;
        }
        return 1;
}
static int initXfiles( i, SideChainsAreSame )
/* reads X file data (reactant descriptors from 2nd comment line of X file ) */
int i, *SideChainsAreSame;
{
        char *foo, *pch;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if (Xfile[i]) {
/* if this X file is same as last, nothing to do */
```

```
            if (!strcmp( Xfile[ i ], foo ) ) return 1;
            *SideChainsAreSame = FALSE;
            UTL_MEM_FREE( Xfile[i] );
          }
          Xfile[ i ] = UTL_STR_SAVE(foo);
          if (! (InputSourceFile = fopen(Xfile[i],"r"))) {
                fprintf( stdout, "Could not open variation file %s\n", Xfile[i] );
                return 0;
          }
/* reading COMMENT lines to get USER_NAME value for matching */
          if ( -1 == UTL_SCAN_GETS( InputSourceFile, "\\", "", &foo)) return 0;
          if ( -1 == UTL_SCAN_GETS( InputSourceFile, "\\", "", &foo)) return 0;
          if (Xname[i]) UTL_MEM_FREE( Xname[i] );
          Xname[i] = 0;
          pch = strstr( foo, "USER_NAME=" );
          pch += strlen( "USER_NAME=" );
          if (!(Xname[i] = UTL_STR_SAVE( pch ) )) return 0;
          fclose( InputSourceFile );
          return 1;
}
int StartFromBitset()
{
   void *CS_PRDCT_BITSET_OPEN();
   if ( !( bitset =  CS_PRDCT_BITSET_OPEN( BitsetFile, MasterRecord))) return 0;

   if ( !RetrieveMasterFileFromBitset(bitset,
                                       &MasterFile,
                                       &MasterRecord, /*in master file*/
                                       0,
                                       0,
                                       0,
                                       0,
                                       0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0,
                                        0  ) ) return 0;
    return 1;
  }
  /*    1/7/97 DEP: allow reading of bitsets. Since the masterfile must be
                  read in any case, the bitset only generates "Dead_Products" */
  int InitMasterFile()
  /* Read the master file record which is requested;
          failure if it does not match the input line info */
  {
   int i, d, size, rxMatch, irx, ns, *Sym;
   char *foo;
   int *fooi;
   if (BitsetFile && ! StartFromBitset()) return 0;
   if (! (MasterFile_File = fopen(MasterFile,"r"))) {
          fprintf( stdout, "%s (master file) not found.\n", MasterFile );
          return 0;
   }
   rxMatch = irx = 0;
   while ( !rxMatch) {
    if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
    if ( strstr(foo,"Reaction class ")) {
          irx++;
```

```
        if (bitset && irx > MasterRecord) return 0; /* the right record did not match */
/* preliminary match if (1) Reaction Needed matches and (2)
          NO_core must be present if NoCore is TRUE (or vice versa) */
        rxMatch = ( irx >= MasterRecord && strstr( foo, ReactionNeeded )
                && ((!NoCore && !strstr( foo, "NO_core" ) )
                 || ( NoCore &&  strstr( foo, "NO_core" ) ) ) );
    }
/* if preliminary match, check rest of .mf record -- first # reactants */
    if (rxMatch) {
/* skip name, record / compare number of reagents */
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if (  ! UTL_STR_ATOI(foo, &d) )                  return 0;
        if (!nR) {
                if (SideChainOnly && d != 1) {
                        fprintf( stdout, "Side Chain only but .mf file references more than
one side chain.\n" );
                        return 0;
                }
                nR = d;
                if (!initXarrays()) return 0;
        }
        rxMatch = nR == d;
    }
    if (rxMatch) {
/* skip fgpt stuff, record core and side chain file stuff */
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if (Corefile) UTL_MEM_FREE( Corefile );
        Corefile = UTL_STR_SAVE(foo);
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if (  ! UTL_STR_ATOI(foo, &StartCore     ) )              return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        for (i = 0; i < nR; i++) if (!initXfiles( i, &SideChainsAreSame ) ) return 0;
    }
  } /* read .mf file until we have a matching reaction */
  return 1;
}
static int ReadXs() {
/* reads all topmer fields from all current Xn files */
  int R, F, i, n, ns, realloc, Fd;
  char *CTOPS, *line, *fptr;
  double *dp, **sdptr;
  unsigned char **uc;
  combi = 1.0;
/* skip the following lengthy stuff if side chains are all the same */
  if (SideChainsAreSame && X[0]) return 1;
  for (R = 0; R < nR; R++) {
    if (! (InputSourceFile = fopen(Xfile[R],"r"))) return 0;
    n = CountLines();
    realloc = n != nX[ R ];
    combi *= (double) n;
    if (realloc && nX[ R ] )
        for (F = 0; F < nFType; F++) {
          for (i = 0; i < nX[R]; i++) UTL_MEM_FREE( *(X[F] + R) + i );
          UTL_MEM_FREE( X[F] + R );
        }

    nX[ R ] = n;
    if (realloc) for (F = 0; F < nFType; F++)
          if (!(*(X[F] + R) = (unsigned char **)
               UTL_MEM_ALLOC( sizeof( unsigned char *) * nX[R]) )) return 0;
 /* starts reading at line 2! */
    for (i = 0; i < nX[R]; i++) {
```

```
    if (-1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &line))
        goto error;
/* generate info for left-to-right read */
    for (F = 0; F < nFType; F++) FOrder[ F ] = strstr( line, FTypes[ F ] );
    do {
      for (Fd = -1, F = 0, fptr = 0; F < nFType; F++)
        if (FOrder[F] && (!fptr || FOrder[F] < fptr)) {fptr = FOrder[F]; Fd =
F;}
      if (fptr) {
        fptr += strlen( FTypes[ Fd ] ) + 1; /*skipping "CTOPS=" */
        UTL_SCAN_TOKENIZE(fptr,';','\\');
        UTL_SCAN_TOKENIZE(fptr,'>','\\');
        if (!ReadAField( fptr, Xsize[ Fd ] + R, *(X[Fd] + R) + i )) goto error;
        FOrder[ Fd ] = 0;
      }
    } while (fptr);
  }
  fclose( InputSourceFile );
/* set up X - Y distance vectors */
  if (realloc) for (F = 0; F < nFType; F++) for (ns = 0; ns < nSym; ns++) {
    sdptr = X2Y[ns];
    if (sdptr[R]) UTL_MEM_FREE( sdptr[R] );
    if (!( sdptr[ R ] = (double *) UTL_MEM_ALLOC( sizeof( double ) * nX[R] ) ))
return 0;
    for (i = 0, dp = sdptr[R]; i < nX[R]; i++) *dp++ = -1.0;
  }
 }
return 1;
error:
 fprintf( stdout, "topsim failed reading line %d of %s.\nLast line read was %s.\n",
    i, Xfile[R], line );
 return 0;
}
```

```
char **ParseQuotedString( SDetails, nDetail, Weights )
char *SDetails;
int *nDetail;
double **Weights;
{
 char *pch, **ppch, *wch, **Details;
 int i;
 double *wt;
/* first trim string to remove leading/trailing spaces and quotes */
   while (*SDetails == '"' || *SDetails == ' ') SDetails++;
   pch = SDetails + strlen( SDetails ) - 1;
   while (*pch == '"' || *pch == ' ') *pch-- = '\0';
/* each space is token delimiter  */
   for (i = 0, pch = SDetails; *pch; pch++)
      if (*pch == ' ') i++;
   *nDetail = i+1;
   if (!(Details = (char **) UTL_MEM_ALLOC( sizeof( char * ) * (*nDetail) ) ))
             return 0;
   if (Weights) {
      if_(!(*Weights = (double *) UTL_MEM_ALLOC( sizeof( double ) * (*nDetail) ) ))
return 0;
      wt = *Weights;
   }
   pch = SDetails;
   if (*pch == '"') pch++;
   for (i = 0, ppch = Details; i < *nDetail; i++, ppch++) {
      UTL_SCAN_TOKENIZE(pch,' ','\\');
      *ppch = UTL_STR_SAVE( pch );
      if (Weights) {
/* note, the copy is now being modified */
        if ((wch = strstr( *ppch, "=" )) ) {
           if (!isweight( wch + 1 )) return FALSE;
           *wt = atof( wch + 1 );
```

```
                *wch = '\0';
            }
            else *wt = 1.0;
            wt++;
        }
        pch += strlen( pch ) + 1;
    }
    return( Details );
}
int isweight( s )
/* returns true if value is a positive decimal value */
char *s ;
{
  char *c;
  for (c = s; *c; c++) if (!isdigit( *c ) && ( *c != '.' )) {
       fprintf( stdout, "Bad weight value: %s. Aborting.\n", s );
       return( FALSE );
  }
  return( TRUE );
}
int ParseRxn()
/* parses complex input descriptions */
{
  char **ParseQuotedString(), **scratch;
  int nRW, i, nX;
  double wtsum;
/* parse field type information or set up standard (steric) type only */
  if (FieldTypes) {
    if (!(FTypes = ParseQuotedString( FieldTypes, &nFType, &FWeights ) )) return 0;
/* scale to average weight of unity */
    for (i = 0, wtsum = 0.0; i < nFType; i++) wtsum += FWeights[i];
    wtsum /= (double) nFType;
    for (i = 0; i < nFType; i++) FWeights[ i ] /= wtsum;
```

```
}
else {
  nFType = 1;
  if (!( FTypes = (char **) UTL_MEM_ALLOC( sizeof( char * ) ) )) return 0;
  if (!( *FTypes = UTL_STR_SAVE( "CTOPS" ) )) return 0;
  if (!( FWeights = (double *) UTL_MEM_ALLOC( sizeof( double ) ) )) return 0;
  *FWeights = 1.0;
}
if (!(FOrder = (char **) UTL_MEM_ALLOC( sizeof(char *) * nFType ) )) return 0;
/* parse any reaction type information present */
nR = 0;
if (SideChainOnly) {
    NoCore = TRUE;
    return 1;
}
if (!ReactionNeeded) return 0;
if (ScratchDetails) {
  if (!(ReactionDetails = ParseQuotedString( ScratchDetails, &nDetail, NIL ) )) return 0;
  nR = nDetail;
  if (!initXarrays()) return 0;
}
if (!(FROrder = (char **) UTL_MEM_ALLOC( sizeof(char *) * nFType * nR ) )) return
0;
/* parse any user-provided variation weighting */
CoreWeight = 1.0;
if (!( RWeights = (double *) UTL_MEM_ALLOC( sizeof( double ) * nR ) )) return 0;
if (XWeights) {
    if (!(scratch = ParseQuotedString( XWeights, &nRW, NIL ) )) return 0;
/* scratch will just be unfreed memory */
    nX = nR + (NoCore ? 0 : 1);
    if (nRW != nX ) {
        fprintf( stdout, "Mismatch between count of xweights (%d) and needed
(%d).\n", nRW, nX );
```

```
            return 0;
        }
        for (i = 0, wtsum = 0.0; i < nR; i++) if (!isweight( scratch[ i ] )) return
FALSE;
            else {
                RWeights[ i ] = atof( scratch[ i ] );
                wtsum += RWeights[ i ];
            }
        if (!NoCore) if (!isweight( scratch[ nR ])) return FALSE;
            else {
                CoreWeight = atof( scratch[ nR ] );
                wtsum += CoreWeight;
            }
        wtsum /= (double) nX;
        for (i = 0; i < nR; i++ ) RWeights[ i ] /= wtsum;
        if (!NoCore) CoreWeight /= wtsum;
  }
  else for (i = 0; i < nR; i++) RWeights[ i ] = 1.0;
  return 1;
}
int ReadEverything()
{
 if (!MasterFile && !BitsetFile) return 0;
 if (!ParseRxn()) return 0;
 setbits_nbits_Init();
 if (!InitMasterFile() ) return 0;
 if (!InitCore() ) return 0;
 if (!WhatsTheDifference()) return 0;
 if (!RetrieveInput() ) return 0;
 return 1;
}
static int InitSym( nsym )
int nsym;
```

```
{
/* sets up symmetries to consider as described for core
   ONLY 2 reactants considered for now!
   assumes that CoreNow is pointing to the appropriate structure */
   int i, F, maxsym;
   double **x2y;
/* get symmetry from current core molecule if not supplied by caller */
   nSym = nsym;
   if ( !nSym ) {
      if ((!strstr( CoreNow, "SYM=" )) || (strstr(CoreNow, "SYM=0")) ) nSym = 1;
      if (strstr(CoreNow, "SYM=1")) nSym = 2;
/* add more categories here */
   }
      for (i = 0; i<nSym; i++) CoreSyms[ i ] = 1;
/* allocate distance arrays to max possible for nR */
   if (!X2Y) {
     for (maxsym = 1, i = 0; i < nR; i++) maxsym *= (i+1);
     if (!(X2Y = (double ***) UTL_MEM_ALLOC( sizeof( double **) * nFType ) ))
 return 0;
     for (i = 0; i < maxsym; i++) {
        if (!(X2Y[i] = (double**) UTL_MEM_ALLOC( sizeof( double *) * nR) )) return 0;
         memset( X2Y[i], 0, sizeof( double *) * nR );
     }
   }
   return nSym;
 }
 int ReadCoreTopomers( CoreOK )
 int *CoreOK;
 {
 /* returns 1 unless fatal error. Sets CoreOK to TRUE if this mf entry is OK
    Also sets up symmetry considerations (which are core structure dependent).
    assumes that CoreNow is pointing to the appropriate structure */
  int foo, i, R, F, Fd, Rd, rf, skipcore, ns, *Sym;
 char label[15], *nxTop, *cstart, *fptr;
 char *cnames[] = {"NX=","NY=","NZ=","CX=","CY=","CZ="};
 double coo;
 double atof();
 skipcore = NoCore;
/* always consider both matches iff no core */
 if(skipcore) InitSym( nR );
        else skipcore = ! InitSym(0);
/* check for any symmetry-allowed rxn by rxn match of all reactant name "details" */
 for (ns = 0; ns < nSym; ns++) if (CoreSyms[ns]) {
        Sym = *(SymList + ns);
        *CoreOK = TRUE;
        if (!SideChainOnly)
          for (i = 0; i < nR && *CoreOK; i++)
           if (!strstr( ReactionDetails[ Sym[ i ] ], Xname[ i ]  ))
                *CoreOK = FALSE;
        if (*CoreOK) break;
 }
 if (skipcore || CoreIsSame || !(*CoreOK )) return 1;
 nxTop = CoreNow;
/* read left-to-right, so record all starting points;
  assume that coords are bunched and appear only once
*/
 for (F = 0; F < nFType; F++) for (R = 0; R < nR; R++) {
    sprintf( label, "%s%d", FTypes[ F ], R + 1 );
    if (!( FROrder[ F * nR + R ] = strstr( nxTop, label ) ) ) {
/* some requested datum missing; then this core entry has no topomer data; use it */
        *CoreOK = 0;
        return 1;
    }
 }
 cstart = strstr( nxTop, cnames[ 0 ] );
 do {
```

```
/* find next datum in left-to-right  rder */
    for (F = 0, fptr = 0; F < nFType; F++) for (R = 0; R < nR; R++) {
        rf = F * nR + R;
        if (FROrder[rf] && (!fptr || FROrder[rf] < fptr)) {fptr = FROrder[rf]; Fd = F;
Rd = R;}
    }
    if (cstart && (!fptr || cstart < fptr)) {fptr = cstart; Fd = -1; }
    if (fptr) {
/* unpack next piece of data to proper location */
        if (Fd >= 0) {
/* then datum is a field */
                fptr += strlen( FTypes[ Fd ] ) + 2; /*skipping "CTOPn=" */
                UTL_SCAN_TOKENIZE(fptr,';','\\');
                UTL_SCAN_TOKENIZE(fptr,'>','\\');
                if (!ReadAField( fptr, Xsize[ Fd ] + Rd, Y[Fd] + Rd )) return 0;
                FROrder[ Fd * nR + Rd ] = 0;
        }
        else {
         for (i = 0; i < 6; i++) {
/* the next data are coordinates */
/* read coords, save as distances squared */
              cstart = strstr( cstart, cnames[i]);
              if (!cstart) {
/* then this core entry has no topomer data */
                  *CoreOK = 0;
                  return 1;
              }
              cstart += strlen(cnames[i]);
              UTL_SCAN_TOKENIZE(cstart,';','\\');
              UTL_SCAN_TOKENIZE(cstart,'>','\\');
              coo = CXcoords[ i ] - atof(cstart);
              CXdiffsq[ i ] = coo * coo * DWeight;
              cstart += strlen( cstart ) + 1;
```

```
            }
            cstart = 0;
          }
      }
} while (fptr);
 return 1;
}
int CoreMatches( CoreOK )
int *CoreOK;
{
/* returns 1 unless fatal error. Sets CoreOK to FALSE if no compound having
        this core can possibly match */
  int F, R, i, ns, *Xct, ct;
  double sqrt(), totd, xount, cdiff;
  unsigned char *ptr, *qtr;
 if (NoCore || CoreIsSame) {
        *CoreOK = TRUE;
        return 1;
 }
/* can check for coordinate discrepancy fast! */
 for (i = 0, cdiff = 0.0; i < 6; i++) cdiff += CXdiffsq[i];
 if (cdiff > Distance) {
        *CoreOK = FALSE;
        return 1;
 }
 for (F = 0, totd = cdiff; F < nFType; F++) for (R = 0; R < nR; R++) {
        if (totd > Distance) break;
        ptr = (unsigned char *) *(Y[ F ] + R);
        qtr = (unsigned char *) *(Yin[F] + R);
        if (!ptr || !qtr) xount = 999999.0;
        else for(xount=0.0, i=0; i < *(Xsize[F] + R); i++, ptr++, qtr++)
          xount += Dist[  *ptr & 0x0F      ][  *qtr & 0x0F      ]
                + Dist[ (*ptr & 0xF0) >> 4][ (*qtr & 0xF0) >> 4]  ;
        totd += xount * FWeights[ F ] / (double) nR;
    }
   CoreDistance = totd * CoreWeight;
   *CoreOK = totd <= Distance;
   return 1;
  }
  int FindXMatches () {
    int R, F, i, ns, ct, *Sym, size,  what ;
    double totd, d, **sdptr, *dptr, xount;
    unsigned char  *ptr, *qtr;
  /* reinitialize indices for permuting over all products  --
          code is general for any number of variable positions */
    for (i = 0; i < nR; i++) Xct[i] = 0;
                   AddressSize(nR, nX, &size);
                   size = (size + 31 )/32 * 4;
   if (bitset) /* assumes actuallsizes matches current sizes!*/
                 {
                    if (!(Dead_Products = (int *) UTL_MEM_ALLOC(size))) return 0;
                    CS_PRDCT_BITSET_TO_RAW( bitset, Dead_Products,  0);
                   not_here(Dead_Products,size  );
                 }

     while ( TRUE ) {
   /* exit elsewhere when all products are enumerated */
              IndexesToAddress( nR, nX, &what, Xct);
            if (Dead_Products &&
                TestDead(0, what) ) goto tupledone;  /* not doing this one! */

            for (ns = 0; ns < nSym; ns++) if (CoreSyms[ns]) {
    /* process all symmetries of current side chain combo */
              Sym = *(SymList + ns);
              sdptr = *(X2Y + ns);
             for (R = 0, totd = CoreDistance;  R < nR; R++) {
```

```
        if (totd > Distance) break;
/* compute next distance if not already done -- DEP knows how this works! */
         dptr = (*(sdptr + R )+Xct[R]);
         if ((*dptr) < 0.0) for (F = 0; F < nFType; F++ ) {
           ptr = (unsigned char *) *( *(X[F] + R) + Xct[ R ]);
           qtr = (unsigned char *)     *(Xin[ F ] + Sym[ R ]) ;
           if (!ptr || !qtr) {*dptr = 999999.0; break;}
           else {
             for(xount=0.0, i=0; i < *(Xsize[F] + R); i++, ptr++, qtr++)
              xount += Dist[  *ptr & 0x0F      ][  *qtr & 0x0F      ]
                     + Dist[ (*ptr & 0xF0) >> 4][ (*qtr & 0xF0) >> 4]  ;
             *dptr += xount * FWeights[ F ];
           }
         }
        totd += *dptr * RWeights[ R ];
      }
/* if hit, write it out */
      if (totd <= Distance) {
        if (NotBitOutput || nR != 2) {
/* ASCII index form of output  -- also REQUIRED if more than 2 varying elements */
             if (!OutputFile && !OpenOutputFile() ) return 0;
             for (R = 0; R < nR; R++) fprintf( OutputFile, "%6d ", Xct[R] + 1 );
             fprintf( OutputFile, "%6d%8.2f%8.2f%8.2f\n", StartCore,
                    sqrt(totd), sqrt(CoreDistance), sqrt(totd - CoreDistance) );
        }
        else {
             if (!Good_Products ) {
                    if (!(Good_Products = (int *) UTL_MEM_ALLOC( size ) )) return
0;
                    memset( Good_Products, 0, size );
             }
             FlagProduct(Good_Products, 0, 0, what );
        }
        nout++;
        if (NoMorehitsPlease && nout >= NoMorehitsPlease) goto done;
/* output only one acceptable symmetry per product */
        goto tupledone;
       }
     }
/* generate next index tuple, AKA candidate product */
tupledone:
     ct = nR - 1;
     while ( TRUE ) {
        Xct[ ct ] ++;
        if (Xct[ ct ] < nX[ ct ]) break;
/* finished when first index exceeds limit -- the other exit  */
        if (ct == 0) goto done;
        Xct[ ct ] = 0;
        ct--;
     }
  }
 done:
 /* output any products from this dataset */
   if (NotBitOutput || nR != 2) {
     if (OutputFile) fclose(OutputFile);
     OutputFile = 0;
   }
   else if (Good_Products) {
        WriteStdFile();
        UTL_MEM_FREE( Good_Products );
        Good_Products = (int*) 0;
   }
   return 1;
 }
 int MakeOutputFileName() {
 /* a run may produce multiple files, and the user probably can't tell,
        so append a sequence _# to subsequent base names */
    if (!nOutFiles) {
        sprintf( OutputFileName, "%s", OutputFileBase );
/* get base name ready for next call */
        strtok( OutputFileBase, "." );
    }
    else sprintf( OutputFileName, "%s_%d.%s", OutputFileBase,
                nOutFiles, OutputFileBase + strlen(OutputFileBase) + 1 );
    nOutFiles++;
}
int WriteStdFile() {
/* writes out the bit set of products */
int sizes[2];
int allocSizes[2] ;
int numInSites[2] ;
void *compressed ;
int  total ;
      sizes[0] = nX[0] ;
      sizes[1] = nX[1] ;
      numInSites[0] = numInSites[1] = -1  ;
      allocSizes[0] = allocSizes[1] = -1 ;
      compressed = NIL;
      total      =   0;
        MakeOutputFileName();
        WriteOutCheckPointFile( OutputFileName,
                MasterFile,
                MasterRecord,
                comline,
                Good_Products,
                0,
                2,
                sizes,
                allocSizes,
            nout,
            numInSites,
            total,
            compressed);
}
int ReadNextCore( SideChainsAreSame, CoreIsSame )
int *SideChainsAreSame;
int *CoreIsSame;
{
/* continues reading through master file for more matching Reaction Classes.
      If the side chain files have the same name, can skip rebuild of X diffs  */
  char *foo;
  int i, d, rxMatch = 0, val;
  if (AllCores) {
      if ( -1 == UTL_SCAN_GETS( CoreFile_File, "\\", "#", &foo)) fclose(
CoreFile_File );
      else {
/* get next core ready and quit */
        CoreNow = UTL_STR_SAVE(foo);
        *SideChainsAreSame = TRUE;
        StartCore++;
        return 1;
      }
  }
  while ( !rxMatch ) {
    if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
/* preliminary match if (1) Reaction Needed matches and (2)
         NO_core must be present if NoCore is TRUE (or vice versa) */
    rxMatch = ( strstr(foo,"Reaction class ") && strstr(foo, ReactionNeeded)
              && ((!NoCore && !strstr( foo, "NO_core" ) )
               || ( NoCore &&  strstr( foo, "NO_core" ) ) ) );
    if (feof(MasterFile_File)) return 0;
/* skip name, record number of reagents */
```

```
    if (rxMatch) {
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if (  ! UTL_STR_ATOI(foo, &val) ) return 0;
        if (val != nR) rxMatch = 0;
    }

    if (rxMatch) {
/* skip fgpt stuff, record core and side chain file stuff */
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        *CoreIsSame = TRUE;
        if (strcmp( foo, Corefile )) {
           *CoreIsSame = FALSE;
           UTL_MEM_FREE( Corefile );
           Corefile = UTL_STR_SAVE(foo);
        }
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if (  ! UTL_STR_ATOI(foo, &val ) )                return 0;
        if (val != StartCore ) *CoreIsSame = FALSE;
        StartCore = val;
        if (! *CoreIsSame ) {
          if (CoreFile_File) fclose(CoreFile_File);
          if (! (CoreFile_File = fopen(Corefile,"r"))) return 0;
          i=0;
          while ( i < StartCore ) {
                if ( -1 == UTL_SCAN_GETS( InputSourceFile, "\\", "#", &foo)) return 0;
                if (AllCores) break;
                i++;
          }
          CoreNow = UTL_STR_SAVE( foo );
        }
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        if ( -1 == UTL_SCAN_GETS( MasterFile_File, "\\", "#", &foo)) return 0;
        *SideChainsAreSame = TRUE;
        for (i = 0; i < nR; i++) if (!initXfiles( i, SideChainsAreSame ) ) return 0;
    }
  }
  return 1;
}
/* this belongs in the utl module, actually */
int MakeComLine( char *line, int len, int argc, char **argv)
{
  int i, nch, totch = 0;
  sprintf(line,"%s ",argv[0]);
  for(i=1;i<argc && totch <= len;i++)
  {
    nch = strlen(line);
    line += nch;
    totch += nch;
    if (totch < len ) sprintf(line,"%s ",argv[i]);
  }
}
int CheckPointProgram(void) {
  fprintf(stderr,"CheckPointProgram() is a lonely stub in topsim.c!\n");
}
int main( argc, argv )
int     argc;
char    **argv;
{
        int processing;
        if( !ParseArguments( argc, argv ) )
                goto SyntaxError;
        MakeComLine( comline, 2048, argc, argv );
        if (!ReadEverything())    goto FailureExit;
        processing = 1;
        while (processing) {
          if (!ReadCoreTopomers( &CoreOK )) goto FailureExit;
          if (CoreOK && !CoreMatches( &CoreOK )) goto FailureExit;
          if (CoreOK && !ReadXs()) goto FailureExit;
          searched += combi;
          if (CoreOK && !FindXMatches()) goto FailureExit;
          totnout += nout;
          nout = 0;
          processing = ReadNextCore( &SideChainsAreSame, &CoreIsSame ) &&
               (!NoMorehitsPlease || nout < NoMorehitsPlease);
        }
        fprintf(stdout, "Normal Exit: %d of %f are neighbors\n", totnout, searched );
        UserAborted ? exit(ErrorExit) : exit(GoodExit);
SyntaxError:
        exit(1);
FailureExit:
        exit(ErrorExit);
}
/*
  numVariations is number of dimensions Y_01, Y_02 etc (normally 2)
  dsize contains the nY_01, nY_02 etc
  address is the bit number (0 to N-1)
  choices will contain the offsets (0 based) of Y_01, Y_02 etc. on return
*/
int AddressToIndexes(int numVariations, int *allPtr, int address, int *chPtr )
{
        for ( chPtr += (numVariations - 1 ), allPtr += (numVariations - 1) ;
             numVariations-- ;
             allPtr--, chPtr-- )
        {
             *chPtr = address % *allPtr;
             address = address / *allPtr;
        }
        return 1;
}

int  IndexesToAddress(int numVariations, int *allPtr, int *address, int *ind)
{
int—i ;
int   indx = 0 ;
        for (i=0;i<numVariations;i++)
          indx += indx * allPtr[i] + ind[i];
        *address = indx;
        return 1 ;
}
int AddressSize(int numVariations, int *allPtr, int *size)
{
  for ( *size = 1 ; --numVariations; allPtr++) *size *= *allPtr;
  return 1;
}

int not_here( what, nbytes )
unsigned char *what;
int nbytes;
{
  for ( ; nbytes; --nbytes) *what++ = ~*what;
  return 1;
}
```

Appendix "T"

```
@macro FragCTOPS ChSp
#
==========================================
=================================
# Entry point for Web-based topomeric search initialization
#
# sets up a set of topomeric searches, by identifying topomer data arising
from
# substructural searching of SLN patterns found in topfrag.tbl to the
# query structure and generating the topomeric data and search command file
entry
# for all resulting fragmentations of the query structure.
#
#   The Query SLN(s) are assumed to be referenced by $CS_QUERY;
#   The file(s) to be searched are referenced by $CS_DATASET (space
separated)
#   The directory where command files are to be written is $CS_TEMPDIR
#   The GUI parameters are to be in $CS_PARAMETERS
#   The name of the output file(s) is to be in $CS_OUTPUT
# read in the data
globalvar CTOP
globalvar ACD!TopInited
localvar fcmdn fcmd tsln dist t base mf mfo nln nxid ferr fern rxids doit
# check the input parameters
    setvar fern %cat( $CS_TEMPDIR "/CSerror.log" )
    setvar ferr %open( $fern "w" )
    setvar flogn %cat( $CS_TEMPDIR "/topfrag.log" )
    setvar flog %open( $flogn "w" )
    setvar fcmdn %cat( $CS_TEMPDIR "/CSCommands.cmd" )
    setvar fcmd %open( $fcmdn "w" )
    if %not( $fcmd )
        %write( $ferr could not open temp file $fcmdn to write ChemSpace search
cmds. Quitting ) >$nulldev
        return
    endif
  for tsln in $CS_QUERY
    if %pos( "." $tsln )
        setvar nogood TRUE
        if %pos( "<" $tsln )
            if %gt( %pos( "." $tsln ) %pos( "<" $tsln ) )
                setvar nogood
            endif
        endif
        if $nogood
            %write( $ferr Topomeric searches require a monomolecular search target.
Quitting ) >$nulldev
            goto error
        endif
    endif
    %write( $flog QUERY: $tsln >$nulldev
    setvar dist %CS_param_parse( distance $CS_PARAMETERS 91.0 )
    if %not( $dist )
        %write( $ferr No topomeric distance provided. Quitting ) >$nulldev
        goto error
    endif
    setvar priority %CS_param_parse( priority $CS_PARAMETERS 3.0 )
    if %not( $priority )
        %write( $ferr No reaction priority provided. Quitting ) >$nulldev
        goto error
    endif
    %write( $flog Fragment Priority: $priority ) >$nulldev
    setvar CTOP[ ONLY1 ] %CS_param_parse( only_subs $CS_PARAMETERS )
    if $CTOP[ ONLY ]
        %write( $flog Matching Side Chain Only ) >$nulldev
    endif
    setvar CTOP[ WEIGHTS ] %CS_param_parse( xweights $CS_PARAMETERS )
    if $CTOP[ WEIGHTS ]
        %write( $flog User Specified Weighting as: $CTOP[ WEIGHTS ] ) >$nulldev
        for w in $CTOP[ WEIGHTS ]
          setvar pats %search2d( $tsln %arg( 1 %set_unpack( $w ) ) NoDup 0 y )
          if %not( $pats )
                %write( $ferr Weighted search for fragment %arg( 1 %set_unpack( $w ) )
not
in $tsln -- can't happen! ) >$nulldev
                goto error
          else
           if %gt( %count( $pats ) 1 )
                %write( $flog NOTE: Multiple hits for weighting fragment %arg( 1
%set_unpack( $w ) ) in $tsln ) >$nulldev
           endif
          endif
        endfor
    endif
    setvar CTOP[ CHBD ] %CS_param_parse( hbonding $CS_PARAMETERS )
    if $CTOP[ CHBD ]
        %write( $flog FIELDS include Hydrogen Bonding with weight of $CTOP[ CHBD ]
)
>$nulldev
    endif
    zap m1 >$nulldev
    %sln_to_mol( m1 $tsln ) >$nulldev
    if %molempty( m1 )
        %write( $ferr SYBYL cannot handle search target (SLN is: $tsln ).
Quitting ) >$nulldev
        goto error
    endif
    setvar t %mol_info( m1 NATOMS )
    FILLVALENCE M1(*) H 1.0 1.5 1.0 1.5 >$nulldev
```

```
   if $CTOP[ ONLY1 ]
       if %neq( %mol_info( m1 NATOMS ) %math( $t + 1 ) )
        %write( $ferr Side chain search but target $tsln has other than one
unfilled valence ) >$nulldev
        goto error
       endif
   else
       if %neq( %mol_info( m1 NATOMS ) $t )
        %write( $ferr Search Target $tsln has unfilled valences. Quitting )
>$nulldev
        goto error
       endif
   endif
  if $CTOP[ ONLY1 ]
# only one side chain to model is a special case
    CTOP!SideChainOnly $fcmd $ferr $flog $dist
  else
# check for custom topomer fragmentation table or selection
    setvar tftabn
    setvar tfrows
    if $CS_TOPFRAG
        setvar t %pos( "_" $CS_TOPFRAG )
        if %not( $t )
            %write( $ferr Custom table name $CS_TOPFRAG missing an "_" ) >$nulldev
            goto error
        else
            setvar tftabn %substr( $CS_TOPFRAG 1 %math( $t - 1 ) )
            setvar tfrows  %substr( $CS_TOPFRAG %math( $t + 1 ) )
        endif
    endif
    if %set_and( "%set_create( %table_name() )" TOPFRAG )
        table close TOPFRAG
    endif
    if %not( $tftabn )
        setvar tftabn %cat( $DSERV_TB topfrag.tbl )
    endif
    table recall $tftabn >$nulldev
    if %not( %set_and( "%set_create( %table_name() )" TOPFRAG ) )
        %write( $ferr $tftabn not found. Quitting ) >$nulldev
        goto error
    endif
    %write( $flog Topomer fragmentation table is %cat( $DSERV_TB topfrag.tbl
) ) >$nulldev
# initialize random file name sequence generator
    setvar t %time()
    setvar base %rand( %substr( "$t" %math( %strlen( "$t" ) - 6 ) 2 ) )
    TAILOR SET MAXIMIN2 MAXIMUM_ITERATIONS 1000 | |
    %write( $flog Master file(s): $CS_DATASET ) >$nulldev
    %write( $flog TOPFRAG table: $tftabn -- Row selection: $tfrows )
>$nulldev
    if %not( $tfrows )
        setvar tfrows %set_create( %range( 1 %table_attribute( NROWS ) ) )
    endif
    for rxid in %set_unpack( $tfrows )
# processing ...
            %write( $flog - - - - - - - - - - - - - - - - - - - - - - ) >$nulldev
# chcek priority
            TABLE Default TOPFRAG
            if %gt( %rcell( $rxid PRIORITY ) $priority )
                %write( $flog TOPFRAG entry $rxid priority > $priority. ) >$nulldev
                break
            endif
            setvar CTOP[RxnCount][$rxid] 0
            if %CS_ReactantMatch( $rxid $fcmd $ferr $tsln $flog )
                    %write( $flog > > > Topomer search queueing (TOPFRAG row $rxid) )
>$nulldev
```

```
                CS!Queue_Search $fcmd $rxid $dist $flog
            endif
       endfor
     endif
   endfor
   # may need to purge or rename error file here!
       %close( $fcmd )
       %close( $ferr )
       %close( $flog )
       return
   error:
       %close( $fcmd )
   # ensure nothing in search command file !
       %file_delete( $fcmdn ) >$nulldev
```

CLAIMS

What is claimed is:

1. A computer-based method for selecting, for all possible product molecules which could be created in a combinatorial synthesis from specified reactant molecules and common core molecule, a subset of product molecules, comprising the following steps:

a. Characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor or to a value close to the neighborhood distance which creates a logical clustering break;

c. Selecting a reactant molecule from each cluster;

d. Combinatorially assembling the selected reactant molecules and core molecule into products which would be created in the chemical synthesis;

e. Selecting a product molecule for inclusion in the subset;

f. Using a validated molecular structural descriptor appropriate to whole molecules, calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Determining the shortest distance between each product molecule and all product molecules previously selected;

h. Selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Repeat steps f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

2. The method of claim 1 in which the validated molecular structural descriptor appropriate to reactant molecules is topomeric CoMFA fields.

3. The method of claim 2 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

4. The method of claim 2 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

5. The method of claim 4 in which before step a, reactant m lecules with the following characteristics are removed from further use in the method:

a. toxic reactant molecules;

b. reactant molecules containing metals, improper forms of tautomers, and interfering chemical groups;

c. reactant molecules with too low a bioavailability;

d. reactant molecules not likely to cross membranes; and

e. reactant molecules containing biologically non-relevant groups.

6. The method of claim 5 in which before step e, product molecules with the following characteristics are removed from further use in the method:

a. product molecules having MW ≥ 750; and

b. product molecules not having a CLOGP between -2 and 7.5.

7. The method of claim 1 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

8. The method of claim 7 in which before step a, reactant molecules with the following characteristics are removed from further use in the method:

a. toxic reactant molecules;

b. reactant molecules containing metals, improper forms of tautomers, and interfering chemical groups;

c. reactant molecules with too low a bioavailability;

d. reactant molecules not likely to cross membranes; and

e. reactant molecules containing biologically non-relevant groups.

9. The method of claim 8 in which before step e, product molecules with the following characteristics are removed from further use in the method:

a. product molecules having MW ≥ 750; and

b. product molecules not having a CLOGP between -2 and 7.5.

10. A computer-based method for selecting, for all possible product molecules which could be created in a combinatorial synthesis from specified reactant molecules, a subset of product molecules, comprising the following steps:

a. Characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor r to a value close to the neighborhood distance which creates a logical clustering break;

c. Selecting a reactant molecule from each cluster;

d. Combinatorially assembling the selected reactant molecules and core molecule into products which would be created in the chemical synthesis;

e. Selecting a product molecule for inclusion in the subset;

f. Using a validated molecular structural descriptor appropriate to whole molecules, calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Determining the shortest distance between each product molecule and all product molecules previously selected;

h. Selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Repeat steps f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

11. The method of claim 10 in which the validated molecular structural descriptor appropriate to reactant molecules is topomeric CoMFA fields.

12. The method of claim 11 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

13. The method of claim 11 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

14. The method of claim 13 in which before step a, reactant molecules with the following characteristics are removed from further use in the method:

a. toxic reactant molecules;

b. reactant molecules containing metals, improper forms of tautomers, and interfering chemical groups;

c. reactant molecules with too low a bioavailability;

d. reactant molecules not likely to cross membranes; and

e. reactant molecules containing biologically non-relevant groups.

15. The method of claim 14 in which before step e, product molecules with the following

characteristics are removed from further use in the method:

a. product molecules having MW ≥ 750; and

b. product molecules not having a CLOGP between -2 and 7.5.

16. The method of claim 10 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

17. The method of claim 16 in which before step a, reactant molecules with the following characteristics are removed from further use in the method:

a. toxic reactant molecules;

b. reactant molecules containing metals, improper forms of tautomers, and interfering chemical groups;

c. reactant molecules with too low a bioavailability;

d. reactant molecules not likely to cross membranes; and

e. reactant molecules containing biologically non-relevant groups.

18. The method of claim 17 in which before step e, product molecules with the following characteristics are removed from further use in the method:

a. product molecules having MW ≥ 750; and

b. product molecules not having a CLOGP between -2 and 7.5.

19. A system for selecting, for all possible product molecules which can be created in a combinatorial synthesis from all specified reactant molecules and common core molecule, a subset of product molecules whose members collectively represent most of the molecular structural diversity in the possible combinatorially synthesized product molecules, comprising:

a. Means for characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Means for hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor or to a value close to the neighborhood distance which creates a logical clustering break;

c. Means for selecting one reactant molecule from each cluster;

d. Means for combinatorially assembling the selected reactant molecules and core molecule into products which would be created in the chemical synthesis;

e. Means for selecting at least one product molecule for inclusion in the subset;

f. Means for using a validated molecular structural descriptor applicable to whole molecules for calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Means for determining the shortest distance between each product molecule and all product molecules previously selected;

h. Means for selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Means for invoking means f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Means for outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

20. The system of claim 19 in which the reactant appropriate molecular structural descriptor is topomeric CoMFA fields.

21. The system of claim 20 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

22. The system of claim 20 in which the whole molecule appropriate molecular structural descriptor is the Tanimoto 2D coefficient.

23. A system for selecting, for all possible product molecules which can be created in a combinatorial synthesis from all specified reactant molecules, a subset of product molecules whose members collectively represent most of the molecular structural diversity in the possible combinatorially synthesized product molecules, comprising:

a. Means for characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Means for hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor or to a value close to the neighborhood distance which creates a logical clustering break;

c. Means for selecting one reactant molecule from each cluster;

d. Means for combinatorially assembling the selected reactant molecules into products which would be created in the chemical synthesis;

e. Means for selecting at least one product molecule for inclusion in the subset;

f. Means for using a validated molecular structural descriptor applicable to whole molecules for calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Means f r determining the shortest distance between each product molecule and all product molecules previously selected;

h. Means for selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Means for invoking means f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Means for outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

24. The system of claim 23 in which the reactant appropriate molecular structural descriptor is topomeric CoMFA fields.

25. The system of claim 24 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

26. The system of claim 24 in which the whole molecule appropriate molecular structural descriptor is the Tanimoto 2D coefficient.

27. A combinatorial screening library designed by a computer-based method, which selects the screening library molecules from those molecules which could be created in a combinatorial synthesis from specified reactant molecules and common core molecule, comprising the following steps:

a. Characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor or to a value close to the neighborhood distance which creates a logical clustering break;

c. Selecting a reactant molecule from each cluster;

d. Combinatorially assembling the selected reactant molecules and core molecule into products which would be created in the chemical synthesis;

e. Selecting a product molecule for inclusion in the subset;

f. Using a validated molecular structural descriptor appropriate to whole molecules, calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Determining the shortest distance between each product molecule and all product molecules previously selected;

h. Selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Repeat steps f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

28. The method of claim 27 in which the validated molecular structural descriptor appropriate to reactant molecules is topomeric CoMFA fields.

29. The method of claim 28 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

30. The method of claim 28 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

31. A combinatorial screening library designed by a computer-based method, which selects the screening library molecules from those molecules which could be created in a combinatorial synthesis from specified reactant molecules, comprising the following steps:

a. Characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor or to a value close to the neighborhood distance which creates a logical clustering break;

c. Selecting a reactant molecule from each cluster;

d. Combinatorially assembling the selected reactant molecules and core molecule into products which would be created in the chemical synthesis;

e. Selecting a product molecule for inclusion in the subset;

f. Using a validated molecular structural descriptor appropriate to whole molecules, calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Determining the shortest distance between each product molecule and all product molecules previously selected;

h. Selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Repeat steps f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

32. The method of claim 31 in which the validated molecular structural descriptor appropriate to reactant molecules is topomeric CoMFA fields.

33. The method of claim 32 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

34. The method of claim 32 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

35. A computer-based method for characterizing the relative validity or usefulness of molecular structural descriptors using multiple literature data sets containing a variety of chemical structures and associated activities comprising the following steps:

a. Applying the molecular structural descriptors to all compounds represented in each data set to derive descriptor values;

b. Constructing a Patterson plot for each molecular structural descriptor for each data set using the descriptor values for the compounds in each data set and their associated activities;

c. Determining the appropriate Patterson plot line and the corresponding density ratio for each molecular structural descriptor for each data set;

d. Determining the number of data sets for each molecular structural descriptor for which the Patterson plots have a density ratio greater than a predetermined cut-off value; and

e. Creating a ranking ratio for each molecular structural descriptor in which the numerator is the number determined in step d and the denominator is the number of data sets, said ranking ratio for each molecular structural descriptor being representative of the relative validity or usefulness of each molecular structural descriptor wherein higher values of the ranking ratio represent a higher degree of validity/usefulness.

36. The method of claim 35 in which in step **d** the predetermined cut-off is about 1.1.

37. A computer-based method f merging with a base assembly of molecules one or more additional assemblies of molecules, similar molecules in the assemblies having previously been identified and removed using a validated molecular structural descriptor, comprising the steps of:

a. Using a validated molecular structural descriptor which is appropriate to whole molecules, characterizing all the molecules in the base assembly of molecules and in the assembly of molecules to be merged;

b. Calculating the molecular structural distance between every molecule in the base assembly to every molecule in the assembly to be merged;

c. While there are still molecules in the assembly to be merged which have not been tested, selecting a molecule from the assembly to be merged;

d. Determining whether the molecular structural distance between the selected molecule and every molecule in the base assembly is within the neighborhood distance of the molecular structural descriptor;

e. Select for inclusion in the merged assemblies only those molecules identified in step **d** as having molecular structural distances greater than the neighborhood distance.

f. Repeat step c through step e until all molecules in the assembly to be merged have been tested; and

g. Repeat step a through step f for each additional assembly to be merged.

38. The method of claim 37 in which the molecular structural descriptor appropriate to whole molecules in the Tanimoto similarity coefficient.

39. A computer-based method of merging with a base assembly of molecules one or more additional assemblies of molecules, similar molecules in one or more of the assemblies having not previously been identified and removed using a validated molecular structural descriptor, comprising the steps of:

a. Selecting subsets of each assembly by:

(1) Selecting a molecule within each assembly;

(2) Using a validated molecular structural descriptor appropriate to whole molecules, calculating the descriptor distance between the selected molecule and all molecules within the assembly;

(3) Determining the shortest distance between the selected molecule and all

molecules previously selected for the subset;

(4) Selecting for inclusion in the subset the molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

(5) Repeat steps (1) through (4) until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

(6) Repeat steps (1) through (5) for each assembly;

b. Using a validated molecular structural descriptor which is appropriate to whole molecules, characterizing all the molecules in the base assembly of molecules and in the assembly of molecules to be merged;

c. Calculating the molecular structural distance between every molecule in the base assembly to every molecule in the assembly to be merged;

d. While there are still molecules in the assembly to be merged which have not been tested, selecting a molecule from the assembly to be merged;

e. Determining whether the molecular structural distance between the selected molecule and every molecule in the base assembly is within the neighborhood distance of the molecular structural descriptor;

f. Select for inclusion in the merged assemblies only those molecules identified in step e as having molecular structural distances greater than the neighborhood distance.

g. Repeat step d through step f until all molecules in the assembly to be merged have been tested; and

h. Repeat step b through step g for each additional assembly to be merged.

40. The use of a subset of molecules, which could be made in a combinatorial synthesis of specified reactants-and core, to specify the compounds to be synthesized and tested in biological screening assays, said subset being selected by the following computer-based method:

a. Characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor or to a value close to the neighborhood distance which creates a logical clustering break;

c. Selecting a reactant molecule from each cluster;

d. Combinatorially assembling the selected reactant molecules and core molecule into products which would be created in the chemical synthesis;

e. Selecting a product molecule for inclusion in the subset;

f. Using a validated molecular structural descriptor appropriate to whole molecules, calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Determining the shortest distance between each product molecule and all product molecules previously selected;

h. Selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Repeat steps f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

41. The method of claim 40 in which the validated molecular structural descriptor appropriate to reactant molecules is topomeric CoMFA fields.

42. The method of claim 41 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

43. The method of claim 41 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

44. The molecules selected, from those which could be made in a combinatorial synthesis of specified reactants and core, by the following computer-based method:

a. Characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor or to a value close to the neighborhood distance which creates a logical clustering break;

c. Selecting a reactant molecule from each cluster;

d. Combinatorially assembling the selected reactant molecules and core molecule into products which would be created in the chemical synthesis;

e. Selecting a product molecule for inclusion in the subset;

f. Using a validated molecular structural descriptor appropriate to whole molecules, calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Determining the shortest distance between each product molecule and all product molecules previously selected;

h. Selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Repeat steps f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

45. The method of claim 44 in which the validated molecular structural descriptor appropriate to reactant molecules is topomeric CoMFA fields.

46. The method of claim 45 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

47. The method of claim 45 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

48. A computer-based method of determining the neighborhood distance characteristic of a validated molecular structural descriptor using multiple literature data sets containing a variety of chemical structures and associated activities, comprising the following steps:

a. Applying the molecular structural descriptor to all compounds represented in each data set to derive descriptor values;

b. Constructing a Patterson plot for each molecular structural descriptor for each data set using the descriptor values for the compounds in each data set and their associated activities;

c. Determining the appropriate Patterson plot line for each data set;

d. Using for each data set a point on the Y axis of the corresponding Patterson plot the end point of an activity difference for which a neighborhood distance is desired, determining the X axis values of the molecular structural descriptor corresponding to the projection from the Patterson plot line of the end points of the activity difference;

e. Determining the average range of values for the neighborhood distance from the plots for each of the data sets.

49. A method of determining the molecules within any set which are most likely to have the same activity as a lead molecule previously identified in an assay comprising the following steps:

a. Characterizing the lead molecule and all other compounds to be examined using a validated molecular structural descriptor appropriate to whole molecules;

b. Determining the molecular structural descriptor distances between the lead molecule and all the other molecules; and

c. Identifying the molecules whose distances from the lead molecule fall within the neighborhood distance of the lead.

50. The method of claim 49 further comprising the additional steps of:

d. Determining the molecular structural descriptor distances between the set of molecules previously identified and all the other molecules excluding the lead and the sets;

e. Identifying the molecules whose distances from molecules in the previously selected set fall within the neighborhood distance; and

f. Repeating steps d through e as many times as desired.

51. A method of determining the useful boundaries of exploration within any set of molecular structures for molecules possessing the same activity as a lead molecule previously identified in an assay comprising the following steps:

a. Characterizing the lead molecule and all other compounds to be examined using a validated molecular structural descriptor appropriate to whole molecules;

b. Determining the molecular structural descriptor distances between the lead molecule and all the other molecules; and

c. Identifying the molecules whose distances from the lead molecule fall within the neighborhood distance of the lead;

d. Synthesizing and testing in an assay the molecules identified in step c and if no activity is detected, stop.

e. If activity is detected, calculating molecular structural descriptor distances, from each molecule identified in the previous step as showing activity, to all other compounds (excluding the lead compound and each previously identified active compound);

f. Identifying all molecules within the neighborhood diameter of the previously identified active molecules;

g. Synthesizing and testing in an assay the molecules identified in the previous step, and if no activity is detected, stop; and

h. Repeating steps e through g until no further compounds show activity in the assay.

52. A computer-based method of characterizing the three dimensional structure of reactants, which can assume many conformations, comprising the steps of:

a. Topomerically aligning the reactants; and

b. Determining the CoMFA steric fields for each topomerically aligned reactant.

53. The method of claim 52 further comprising the addition of topomeric hydrogen bonding fields to the CoMFA steric fields.

54. A computer-based method of applying a molecular structural descriptor to a set of reactants comprising the following steps:

a. Topomerically aligning the reactants;

b. Determining the CoMFA steric fields for each topomerically aligned reactant; and

c. Calculating the field differences between all pairs of reactants.

55. The method of claim 54 further comprising after step b the additional step of adding topomeric hydrogen bonding fields to the CoMFA fields.

56. The method of claim 54 further comprising after step c the additional step of hierarchically clustering the reactants until the intercluster distance is about 80 - 100 CoMFA field units.

57. In a digital computer in which representations of specified reactant molecules and a core molecule have been stored, a computer-based method for selecting, for all possible product molecules which could be created in a combinatorial synthesis from the reactant molecules and common core molecule, a subset of product molecules, comprising the following steps:

a. Characterizing all the reactant molecules with a validated molecular structural descriptor appropriate to reactant molecules;

b. Hierarchically clustering the characterized reactant molecules until the intercluster distance corresponds to the neighborhood distance of the validated molecular structural descriptor or to a value close to the neighborhood distance which creates a logical clustering break;

c. Selecting a reactant molecule from each cluster;

d. Combinatorially assembling the selected reactant molecules and core molecule into products which would be created in the chemical synthesis;

e. Selecting a product molecule for inclusion in the subset;

f. Using a validated molecular structural descriptor appropriate to whole molecules, calculating the descriptor distance between all selected product molecules and all other product molecules;

g. Determining the shortest distance between each product molecule and all product molecules previously selected;

h. Selecting for inclusion in the subset the product molecule whose shortest descriptor distance from the previously selected molecules is the largest and is greater than the neighborhood distance of the descriptor;

i. Repeat steps f through h until the largest shortest difference between molecules is less than the neighborhood distance of the descriptor; and

j. Outputing a list of the selected product molecules and/or the reactant molecules from which the selected product molecules can be formed.

58. The method of claim 57 in which the validated molecular structural descriptor appropriate to reactant molecules is topomeric CoMFA fields.

59. The method of claim 58 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

60. The method of claim 57 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

61. A computer-based method for generating a virtual library of possible combinatorially derived product molecules which can be searched for product molecules having desired properties without the necessity of generating the product structures during the search, comprising the following steps:

a. Creating one or more files identifying one or more combinatorial reactions for one or more core structures;

b. Creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

c. Associating with each structural variation, data, characterizing each structural variation including:

(1) Characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(2) Characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations.

62. A virtual library of possible combinatorially derived product molecules which can be searched for product molecules having desired properties without the necessity of generating the product structures during the search, generated by the following process:

a. Creating one or more files identifying one or more combinatorial reactions for one or more core structures;

b. Creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

c. Associating with each structural variation, data, characterizing each structural variation including:

(1) Characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(2) Characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations.

63. The method of claim 61 further comprising a computer-based method for selecting from the virtual library, for all possible product molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule, a subset of product molecules, comprising the following additional steps:

b. identifying all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. using a validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

e. using a validated molecular descriptor appropriate to the structural variations with which the Virtual Library was generated, removing from the set of all remaining product molecules those molecules formed from structural variations falling within a chosen neighborhood distance of the structural variations of the selected molecule;

f. selecting from the set of all product molecules remaining after step e a product molecule for inclusion in the subset;

g. repeating steps d through f until no additional product molecules remain to be selected in step f; and

h. Outputting a list of the selected subset and/or the structural variations from which the subset can be formed.

64. The method of claim 61 further comprising a computer-based method for selecting from the virtual library, for all possible product molecules which could be created by all combinatorial arrangements of specified structural variations and core molecules, a subset of product molecules, comprising the following additional steps:

b. selecting from all possible cores a core upon which to base the subset;

c. using a validated molecular descriptor appropriate to cores, selecting from the set of all possible cores those core molecules falling within the neighborhood distance of the selected core molecule;

d. identifying all possible combinatorial product molecules which could result from the specified structural variations and selected core molecules;

e. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

f. using a validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

g. using a validated molecular descriptor appropriate to the structural variations with which the Virtual Library was generated, removing from the set of all remaining product molecules those molecules formed from structural variations falling within a chosen neighborhood distance of the structural variations of the selected molecule;

h. selecting from the set of all product molecules remaining after step g a product molecule for inclusion in the subset;

i. repeating steps f through h until no additional product molecules remain to be selected in step h; and

j. Outputting a list of the selected subset and/or the structural variations and cores from which the subset can be formed.

65. The method of claim 61 further comprising a computer-based method for selecting from the virtual library, for all possible product molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule, a subset of product molecules, comprising the following additional steps:

b. identifying all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. using a validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within the neighborhood distance of the selected molecule;

e. selecting from the set of all product molecules remaining after step **d** a product molecule for inclusion in the subset;

f. repeating steps **d** through **e** until no additional product molecules remain to be selected in step **f**; and

g. Ouputting a list of the selected subset and/or the structural variations from which the subset can be formed.

66. The method of claim 61 further comprising a computer-based method for selecting from the virtual library, for all possible product molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule, a subset of product molecules, comprising the following additional steps:

b. identifying all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. using a validated molecular descriptor appropriate to the structural variations with which the Virtual Library was generated, removing from the set of all remaining product molecules those molecules formed from structural variations falling within a chosen neighborhood distance of the structural variations of the selected molecule;

e. selecting from the set of all product molecules remaining after step **d** a product molecule for inclusion in the subset;

f. repeating steps d through e until no additional product molecules remain to be selected in step e; and

g. Ouputting a list of th selected subset and/or the structural variations from which the subset can be formed.

67. A screening library designed by a computer-based method which selects the screening library molecules from those molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule comprising the following steps:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. using a validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

e. using a validated molecular descriptor appropriate to the structural variations with which the Virtual Library was generated, removing from the set of all remaining product molecules those molecules formed from structural variations falling within a chosen neighborhood distance of the structural variations of the selected molecule;

f. selecting from the set of all product molecules remaining after step e a product molecule for inclusion in the subset;

g. repeating steps d through f until no additional product molecules remain to be selected in step f; and

h. Outputting a list-of-the selected subset and/or the structural variations from which the subset can be formed.

68. A screening library designed by a computer-based method which selects the screening library molecules from those molecules which could be created by all combinatorial arrangements of specified structural variations and core molecules comprising the following steps:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. selecting from all possible cores a core upon which to base the subset;

c. using a validated molecular descriptor appropriate to cores, selecting from the set of all possible cores those core molecules falling within the neighborhood distance of the selected core molecule;

d. identifying all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

e. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

f. using a validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

g. using a validated molecular descriptor appropriate to the structural variations with which the Virtual Library was generated, removing from the set of all remaining product molecules those molecules formed from structural variations falling within a chosen neighborhood distance of the structural variations of the selected molecule;

h. selecting from the set of all product molecules remaining after step g a product molecule for inclusion in the subset;

i. repeating steps f through h until no additional product molecules remain to be selected in step h; and

j. Outputting a list of the selected subset and/or the structural variations and cores from which the subset can be formed.

69. The use of a subset of molecules, which could be made in a combinatorial synthesis of specified reactants and common core, to specify the compounds to be synthesized and tested in appropriate assays, said subset being selected by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. -using a validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

e. using a validated molecular descriptor appropriate to the structural variations with which the Virtual Library was generated, removing from the set of all remaining product molecules those molecules formed from structural variations falling within a chosen neighborhood distance of the structural variations of the selected molecule;

f. selecting from the set of all product molecules remaining after step e a product molecule for inclusion in the subset;

g. repeating steps d through f until no additional product molecules remain to be selected in step f; and

h. Outputting a list of the selected subset and/or the reactants from which the subset can be formed.

70. The molecules selected, from those which could be made in a combinatorial synthesis of specified reactants and common core, by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and (b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and core molecule;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. using a validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

e. using a validated molecular descriptor appropriate to the structural variations with which the Virtual Library was generated, removing from the set of all remaining product molecules those molecules formed from structural variations falling within a chosen neighborhood distance of the structural variations of the selected molecule;

f. selecting from the set of all product molecules remaining after step e a product molecule for inclusion in the subset;

g. repeating steps d through f until no additional product molecules remain to be selected in step f; and

h. Outputting a list of the selected subset and/or the reactants from which the subset can be formed.

71. The molecules selected, from those which could be made in a combinatorial synthesis of specified reactants and cores, by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

-b. selecting from all possible cores a core upon which to base the subset;

c. using a validated molecular descriptor appropriate to cores, selecting from the set of all possible cores those core molecules falling within the neighborhood distance of the selected core molecule;

d. identifying all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

e. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

f. using a validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

g. using a validated molecular descriptor appropriate to the structural variations with which the Virtual Library was generated, removing from the set of all remaining product molecules those molecules formed from structural variations falling within a chosen neighborhood distance of the structural variations of the selected molecule;

h. selecting from the set of all product molecules remaining after step **g** a product molecule for inclusion in the subset;

i. repeating steps **f** through **h** until no additional product molecules remain to be selected in step **h**; and

j. Outputting a list of the selected subset and/or the reactants from which the subset can be formed.

72. The method of claim 1 further comprising a computer-based method for selecting from the virtual library, for all possible product molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule, a subset of product molecules, comprising the following additional steps:

b. identifying all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. using a combination validated molecular descriptor characterizing both whole molecule and structural variation features with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

e. selecting from the set of all product molecules remaining after step d a product molecule for inclusion in the subset;

f. repeating steps d through e until no additional product molecules remain to be selected in step e; and

h. Outputting a list of the selected subset and/or the structural variations from which the subset can be formed.

73. The method of claim 61 further comprising a computer-based method for selecting from the virtual library, for all possible product molecules which could be created by all combinatorial arrangements of specified structural variations and core molecules, a subset of product molecules, comprising the following additional steps:

b. selecting from all possible cores a core upon which to base the subset;

c. using a validated molecular descriptor appropriate to cores, selecting from the set of all possible cores those core molecules falling within the neighborhood distance of the selected core molecule;

d. identifying all possible combinatorial product molecules which could result from the specified structural variations and selected core molecules;

e. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

f. using a combination validated molecular descriptor characterizing both whole molecule and structural variation features with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

g. selecting from the set of all product molecules remaining after step e a product molecule for inclusion in the subset;

f. repeating steps e through g until no additional product molecules remain to be selected in step g; and

h. Outputting a list of the selected subset and/or the structural variations and cores from which the subset can be formed.

74. The molecules selected, from those which could be made in a combinatorial synthesis of specified reactants and common core, by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). -creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and core molecule;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. using a combination validated molecular descriptor characterizing both whole molecule and structural variation features with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

e. selecting from the set of all product molecules remaining after step d a product molecule for inclusion in the subset;

f. repeating steps d through e until no additional product molecules remain to be selected in step e; and

h. Outputting a list of the selected subset and/or the reactants from which the subset can be formed.

75. The molecules selected, from those which could be made in a combinatorial synthesis of specified reactants and cores, by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. selecting from all possible cores a core upon which to base the subset;

c. using a validated molecular descriptor appropriate to cores, selecting from the set of all possible cores those core molecules falling within the neighborhood distance of the selected core molecule;

d. identifying all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

e. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

f. using a combination validated molecular descriptor characterizing both whole molecule and structural variation features with which the Virtual Library was generated, removing from the set of all remaining molecules those molecules falling within a chosen neighborhood distance of the selected molecule;

g. selecting from the set of all product molecules remaining after step f a product molecule for inclusion in the subset;

f. repeating steps f through g until no additional product molecules remain to be selected in step g; and

h. Outputting a list of the selected subset and/or the reactants and cores from which the subset can be formed.

76. The method of claim 61 further comprising a method of determining within the virtual library, the molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule, which are most likely to have the same type of activity as a molecule of interest comprising the following steps:

a. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

b. characterizing the molecule of interest with a validated molecular structural descriptor appropriate to whole molecules with which the virtual library was generated;

d. using the same validated molecular descriptor appropriate to whole molecules, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

g. Ouputting a list of the selected subset and/or the structural variations from which the subset can be formed.

77. The method of claim 61 further comprising a method of determining within the virtual library, the molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule, which are most likely to have the same type of activity as a molecule of interest comprising the following steps:

a. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

b. characterizing the molecule of interest with a validated molecular structural descriptor appropriate to structural variations with which the virtual library was generated;

d. using the same validated molecular descriptor appropriate to structural variations, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

g. Ouputting a list of the selected subset and/or the structural variations from which the subset can be formed.

78. The method of claim 61 further comprising a method of determining within the virtual library, the molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule, which are most likely to have the same type of activity as a molecule of interest comprising the following steps:

a. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

b. characterizing the molecule of interest with both a validated molecular structural descriptor appropriate to structural variations with which the virtual library was generated and with a validated molecular structural descriptor appropriate to structural variations with which the virtual library was generated;

d. using the same validated molecular descriptor appropriate to whole molecules, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule, and using the same validated molecular descriptor appropriate to structural variations, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

e. Ouputting a list of the selected subset and/or the structural variations from which the subset can be formed. .

79. The method of claim 61 further comprising a method of determining within the virtual library, the molecules which could be created by all combinatorial arrangements of specified structural variations and a common core molecule, which are most likely to have the same type of activity as a molecule of interest comprising the following steps:

a. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

b. characterizing the molecule of interest with a combination validated molecular descriptor, characterizing both whole molecule and structural variation features, with which the Virtual Library was generated;

d. using the same validated molecular descriptor, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

g. Ouputting a list of the selected subset and/or the structural variations from which the subset can be formed.

80. The molecules, which are most likely to have the same type of activity as a molecule of interest, selected, from those which could be made in a combinatorial synthesis from specified reactants and a common core molecule, by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. characterizing the molecule of interest with both a validated molecular structural descriptor appropriate to structural variations with which the virtual library was generated and with a validated molecular structural descriptor appropriate to structural variations with which the virtual library was generated;

d. using the same validated molecular descriptor appropriate to whole molecules, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule, and using the same validated molecular descriptor appropriate to structural variations, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

e. Ouputting a list of the selected subset and/or the reactants from which the subset can be formed.

81. The molecules, which are most likely to have the same type of activity as a molecule of interest, selected, from those which could be made in a combinatorial synthesis from specified reactants and a common core molecule, by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. characterizing the molecule of interest with a combination validated molecular descriptor, characterizing both whole molecule and structural variation features, with which the Virtual Library was generated;

d. using the same validated molecular descriptor, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

e. Ouputting a list of the selected subset and/or the reactant from which the subset of molecules can be formed.

82. The use of a subset of molecules, which are most likely to have the same type of activity as a molecule of interest and selected from those which could be made in a combinatorial synthesis from specified reactants and a common core molecule, to specify the compounds to be synthesized and tested in appropriate assays, said subset being selected by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which-has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. characterizing the molecule of interest with both a validated molecular structural descriptor appropriate to whole molecules with which the virtual library was generated and with a validated molecular structural descriptor appropriate to structural variations with which the virtual library was generated;

e. using the same validated molecular descriptor appropriate to whole molecules, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule, and using the same validated molecular descriptor appropriate to structural variations, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

f. Ouputting a list of the selected subset and/or the reactants from which the subset can be formed.

83. The use of a subset of molecules, which are most likely to have the same type of activity as a molecule of interest and selected from those which could be made in a combinatorial synthesis from specified reactants and a common core molecule, to specify the compounds to be synthesized and tested in appropriate assays, said subset being selected by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. identifying in the virtual library all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

c. selecting from all possible combinatorial product molecules a product molecule for inclusion in the subset;

d. characterizing the molecule of interest with a combination validated molecular descriptor, characterizing both whole molecule and structural variation features, with which the Virtual Library was generated;

e. using the same validated molecular descriptor, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

f. Ouputting a list of the selected subset and/or the reactant from which the subset of molecules can be formed.

84. The method of claim 61 further comprising a method of determining within the virtual library, the molecules which could be created by all combinatorial arrangements of specified structural variations and core molecules, which are most likely to have the same type of activity as a molecule of interest, comprising the following steps:

a. selecting from all possible cores a core upon which to base the subset;

b. using a validated molecular descriptor appropriate to cores, selecting from the set of all possible cores those core molecules falling within the neighborhood distance of the selected core molecule;

c. identifying all possible combinatorial product molecules which could result from the specified reactants and selected core molecules;

d. selecting and characterizing the molecule of interest with a validated molecular structural descriptor-appropriate to whole molecules with which the virtual library was generated;

e. using the same validated molecular descriptor appropriate to whole molecules, selecting the set of all possible molecules whose descriptor values fall within a chosen neighborhood distance of the selected molecule; and

f. Ouputting a list of the selected subset and/or the structural variations from which the subset can be formed.

85. The method of claim 61 further comprising a method of determining within the virtual library, the molecules which could be created by all combinatorial arrangements of structural variations and core molecules, which are most likely to have the same type of activity as a molecule of interest, which is not known to be derived from a combinatorial reaction, comprising the following steps:

a. fragmenting the molecule of interest as described in a fragmentation table;

b. selecting a fragmentation pattern;

c. aligning the fragments according to topomeric alignment rules;

d. generating CoMFA fields for each aligned fragment;

e. identifying which reaction types within the virtual library correspond to the reaction type resulting from the fragmentation;

f. identifying whether the fragmentation pattern generated a core, and, if so, implementing the following steps:

(1) characterizing the core with CoMFA fields; and

(2) identifying, by comparing the field values, whether the core resembles any cores used in the creation of the virtual library;

g. selecting structural variations which were used in generating the virtual library with cores which matched the core resulting from the fragmentation;

h. comparing the CoMFA fields of the topomerically aligned fragments with the fields of the identified structural variations by taking the root sum of squares field differences;

i. selecting those structural variations for which the root sum of squares field difference falls within a ch sen neighborhood value;

j. ouputting a list f the selected subset and/or the structural variations from which the subset can be forme;

k. repeating steps b through j for all possible fragments.

86. The molecules, which are most likely to have the same type of activity as a molecule of interest which is not known to be derived from a combinatorial reaction, selected from those product molecules which could be created by all combinatorial arrangements of structural variations and core molecules, by the following computer-based method:

a. generating a virtual library by:

(1). creating one or more files identifying one or more combinatorial reactions for one or more core structures;

(2). creating separate structural variation files (associated with the reaction identifying files) in which are listed together the structural variations representative of those reactants which will react at each variation site of each combinatorial reaction;

(3). associating with each structural variation, data, characterizing each structural variation including:

(a). characterization data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has not been derived from the application of validated molecular structural descriptors; and

(b). characterizing data, taking into account when necessary the structures of the cores with which the structural variations would be combined in the listed combinatorial syntheses, which has been derived from applying validated molecular structural descriptors to the structural variations;

b. fragmenting the molecule of interest as described in a fragmentation table;

c. selecting a fragmentation pattern;

d. aligning the fragments according to topomeric alignment rules;

e. generating CoMFA fields for each aligned fragment;

f. identifying which reaction types within the virtual library correspond to the reaction type resulting from the fragmentation;

g. identifying whether the fragmentation pattern generated a core, and, if so, implementing the following steps:

(1) characterizing the core with CoMFA fields; and

(2) identifying, by comparing the field values, whether the core resembles any cores used in the creation of the virtual library;

h. selecting structural variations which were used in generating the virtual library with cores which matched the core resulting from the fragmentation;

i. comparing the CoMFA fields of the topomerically aligned fragments with the fields of the identified structural variations by taking the root sum of squares field differences;

j. selecting those structural variations for which the root sum of squares field difference falls within a chosen neighborhood value;

k. ouputting a list of the selected subset and/or the structural variations from which the subset can be forme;

l. repeating steps c through k for all possible fragments.

87. The method of claims 63 or 65 or 69 or 71 or 72 or 73 or 74 or 75 or 80 or 86 or 88 in which the following additional step is performed immediately after the step of using a validated molecular descriptor appropriate to whole molecules:

t. repeating the previous step for another validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated until no additional whole molecule descriptor remains to be used.

88. The method of claims 63 or 65 or 70 or 71 or 72 or 73 or 74 or 75 or 81 or 86 in which the following additional step is performed immediately after the step of using a validated molecular descriptor appropriate to structural variations:

u. repeating the previous step for another validated molecular descriptor appropriate to structural variations with which the Virtual Library was generated until no additional structural variation descriptor remains to be used.

89. The method of claim 63 in which the additional step t is performed immediately after the step of using a validated molecular descriptor appropriate to whole molecules and further in which step u is performed immediately after the step of using a validated molecular descriptor appropriate to structural variations:

t. repeating the previous step for another validated molecular descriptor appropriate to whole molecules with which the Virtual Library was generated until no additional whole molecule descriptor remains to be used; and

u. repeating the previous step for another validated molecular descriptor appropriate to structural variations with which the Virtual Library was generated until no additional structural variation descriptor remains to be used.

90. The method of claims 61 or 63 r 65 or 70 or 71 or 72 or 73 or 74 or 86 in which the validated molecular structural descriptor appropriate to structural variations is topomeric CoMFA fields.

91. The method of claim 61 or 63 or 65 or 70 or 71 or 72 or 73 or 74 or 86 in which topomeric hydrogen bond fields are used in conjunction with the topomeric CoMFA fields descriptor.

92. The method of claims 63 or 65 or 69 or 71 or 72 or 73 or 74 or 75 or 80 or 86 or 88 in which the validated molecular structural descriptor appropriate to whole molecules is the Tanimoto 2D coefficient.

93. The method of claim 63 in which after step g product molecules with the following characteristics are removed from further use in the method:

a. toxic reactant molecules;

b. reactant molecules containing metals, improper forms of tautomers, and interfering chemical groups;

c. reactant molecules with too low a bioavailability;

d. reactant molecules not likely to cross membranes; and

e. reactant molecules containing biologically non-relevant groups.

94. The method of claim 63 in which after step g product molecules with the following characteristics are removed from further use in the method:

a. product molecules having MW ≥ 750; and

b. product molecules not having a CLOGP between -2 and 7.5.

95. The methods of selecting screening libraries as disclosed in this invention.

96. The systems for selecting screening libraries as disclosed in this invention.

97. The screening libraries selected by the methods or systems disclosed in this invention.

98. The metric validation method as disclosed in this invention.

99. The method of merging libraries as disclosed in this invention.

100. The method of lead explosion as disclosed in this invention.

101. The methods of molecular alignment as disclosed in this invention.

102. The new molecular structural descriptors as disclosed in this invention.

103. The methods of generating a virtual library as disclosed in this invention.

104. The methods of searching a virtual library as disclosed in this invention.

105. The virtual library as disclosed in this invention.

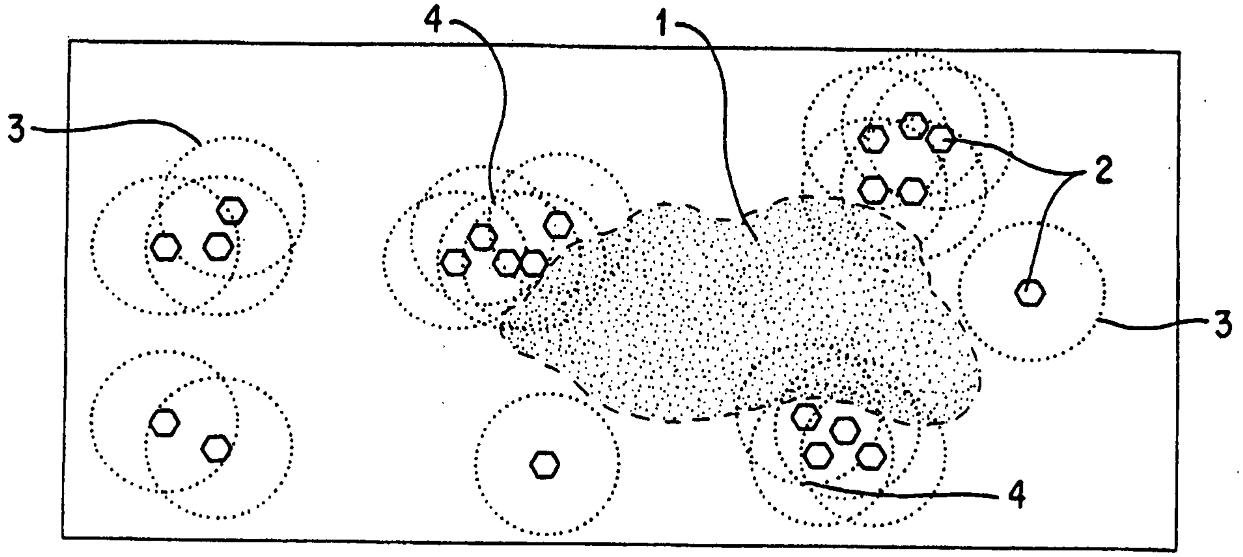


FIG. 1(a)

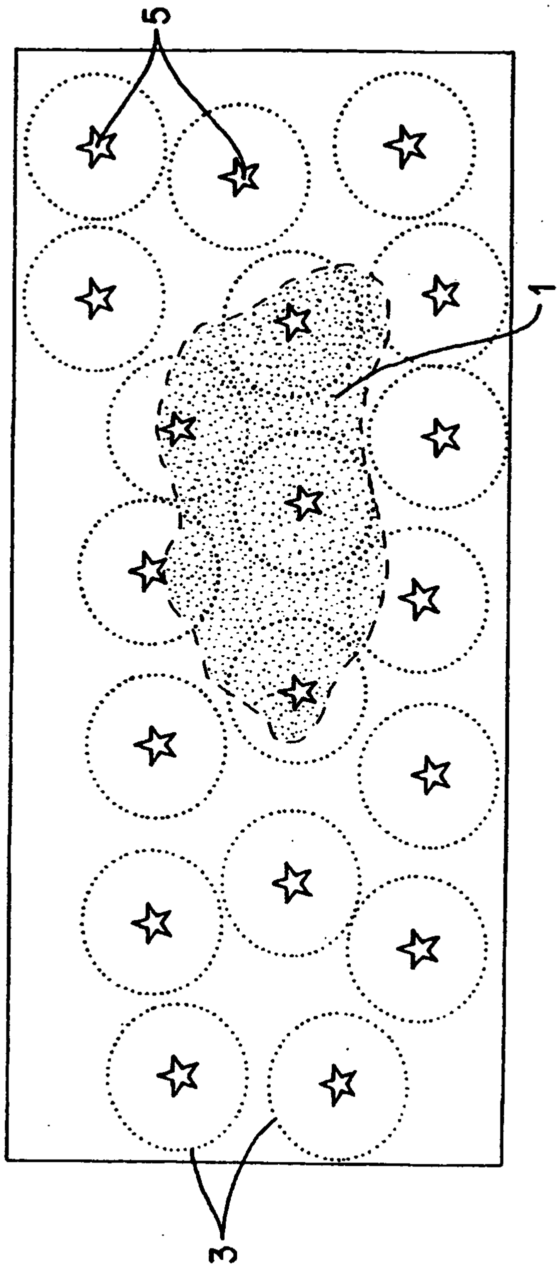


FIG.1(b)

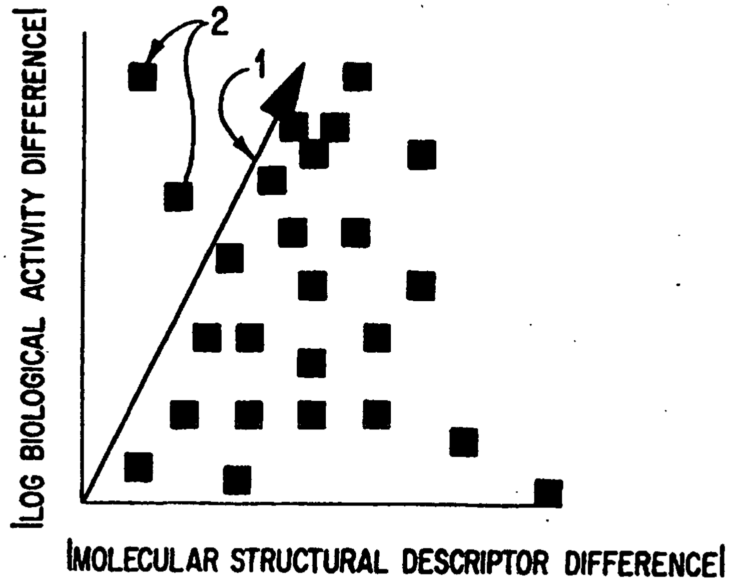
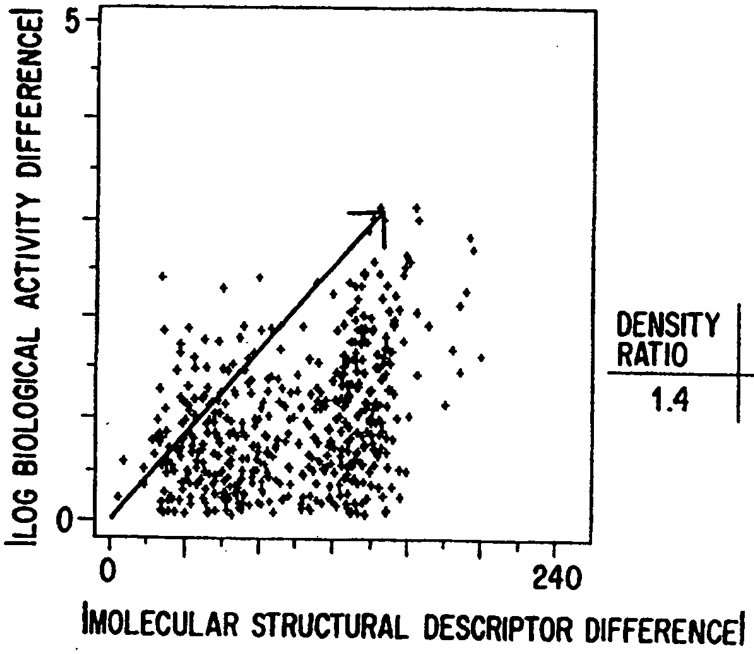
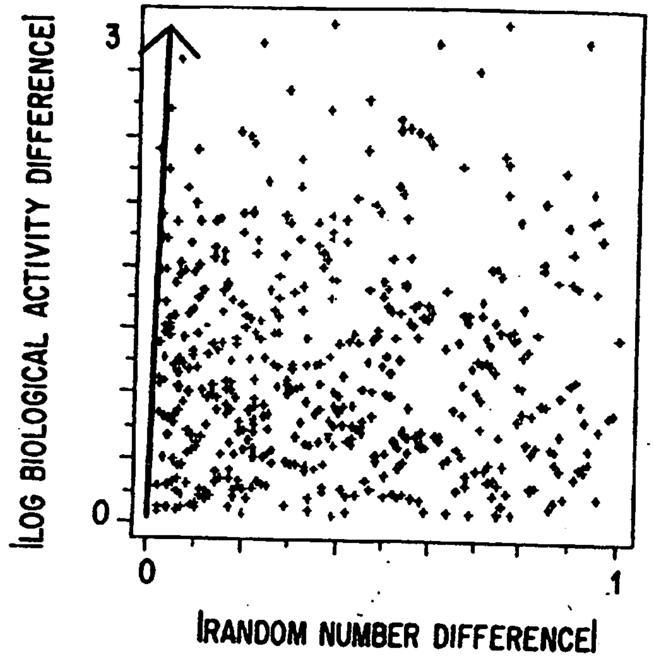


FIG. 2



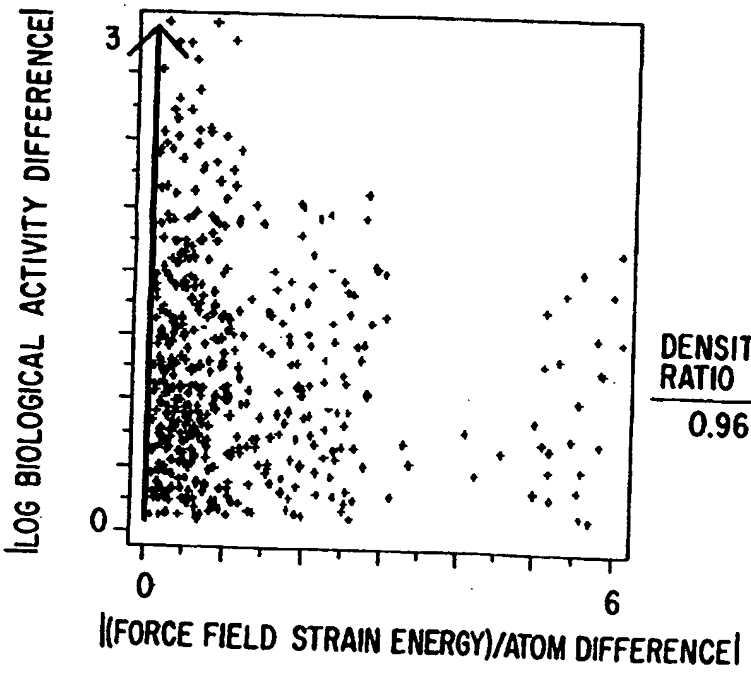
DENSITY RATIO	χ^2
1.4	51.21

FIG. 3



DENSITY RATIO	χ^2
1.00	0.00

FIG. 4



DENSITY RATIO	χ^2
0.96	0.46

FIG. 5

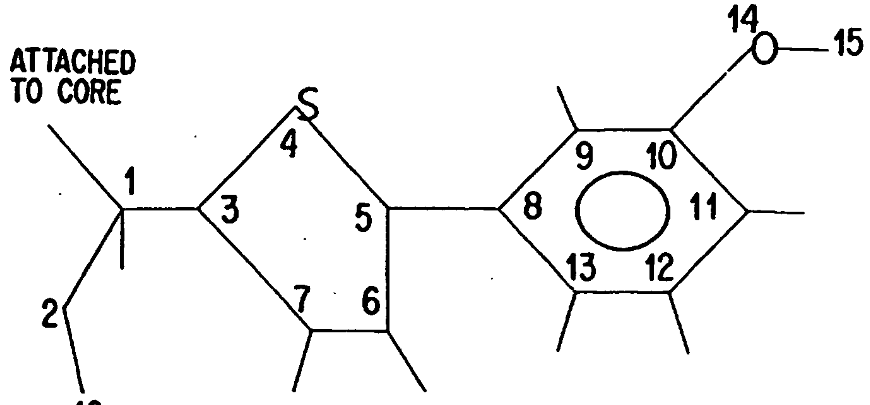


FIG. 6(a)

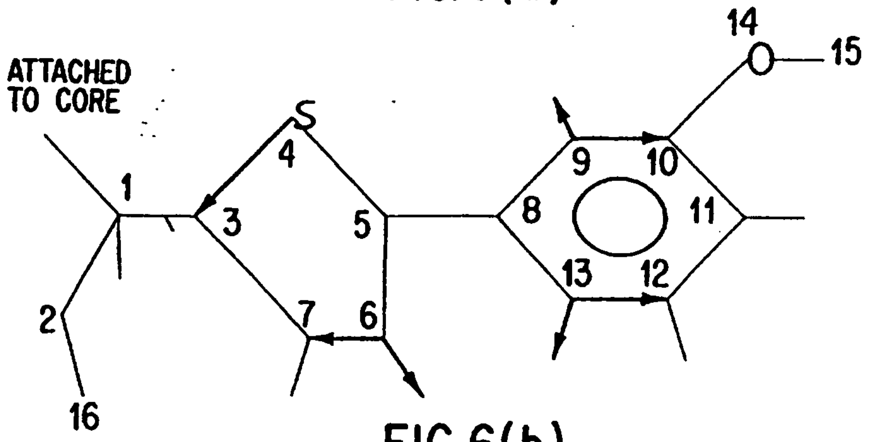


FIG. 6(b)

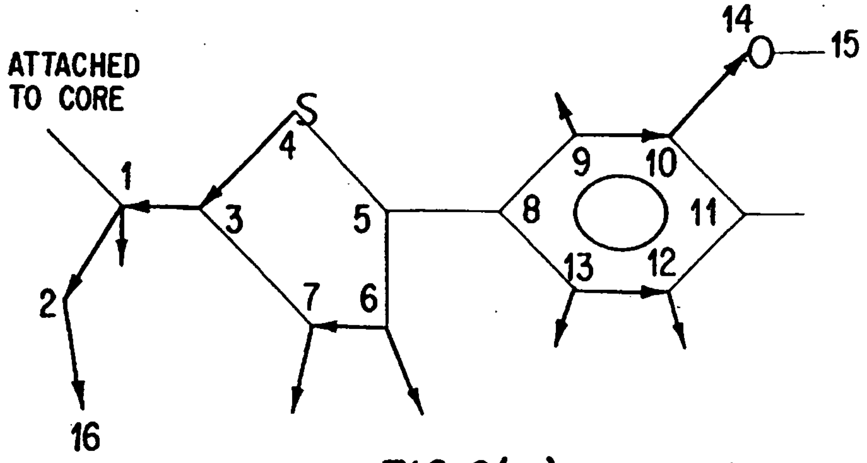


FIG. 6(c)

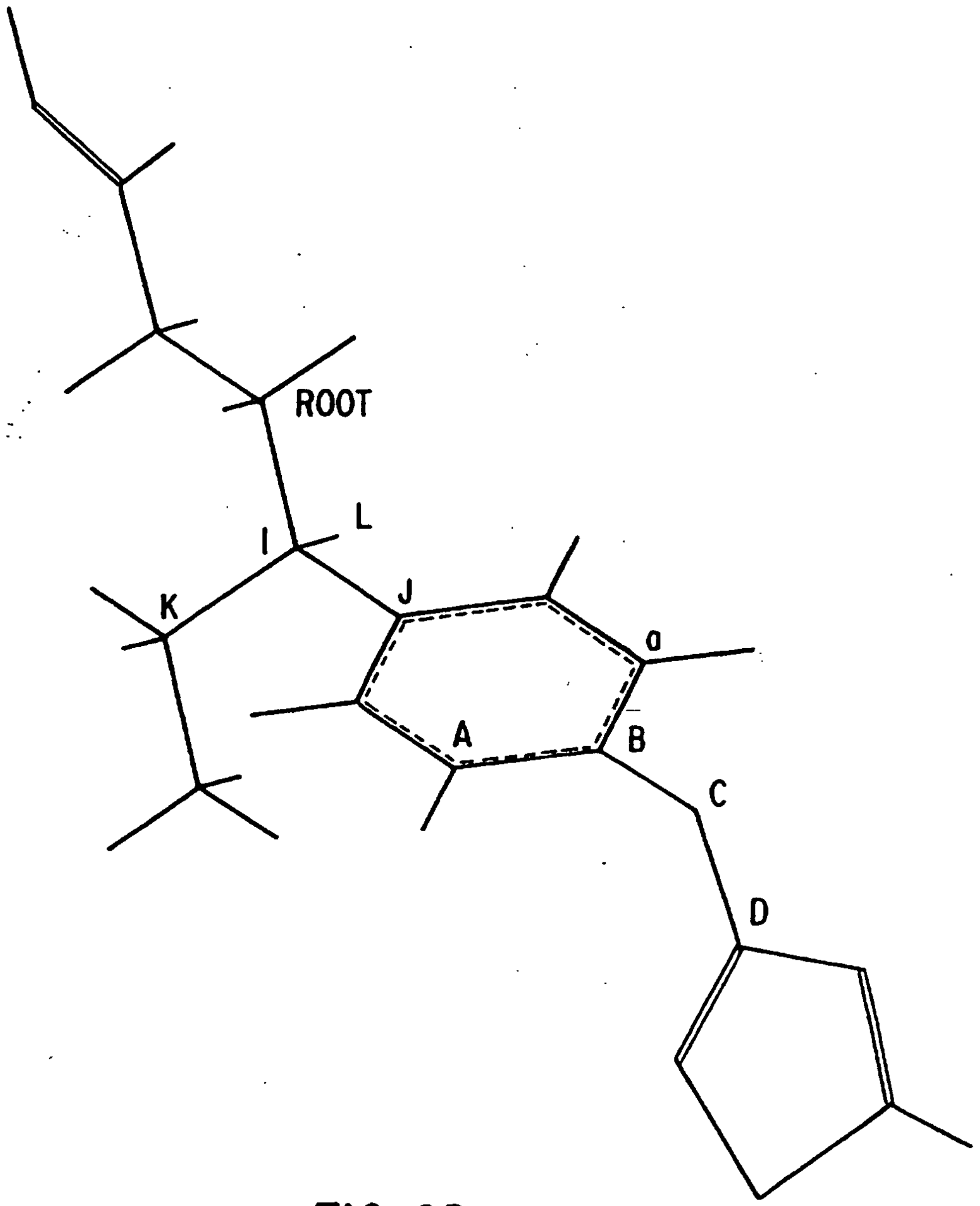


FIG. 6D

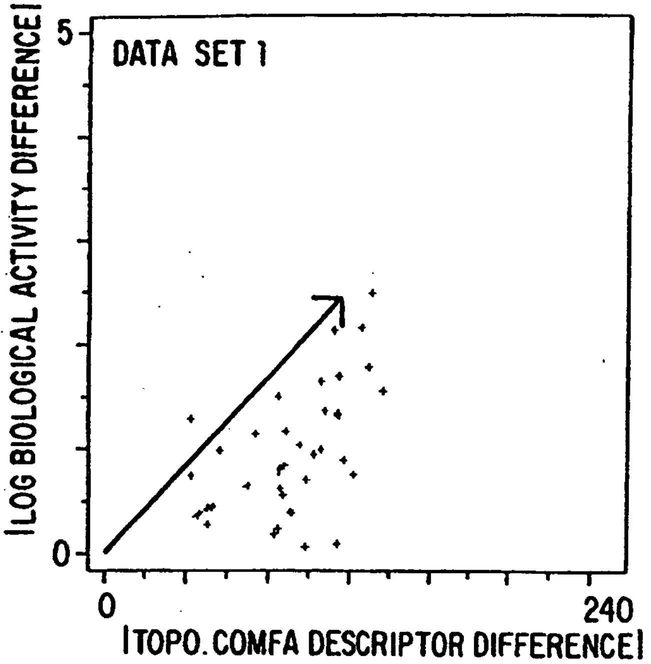

DATA SET 1
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(a)

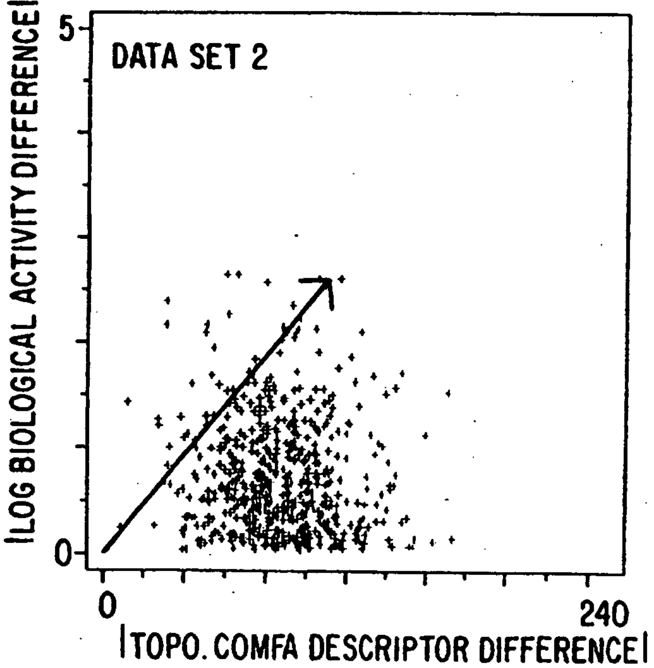

DATA SET 2
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(b)

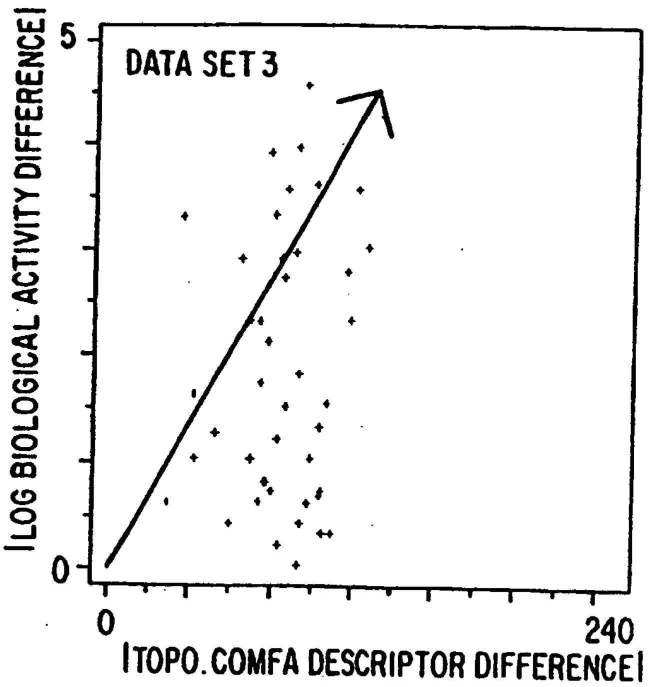

DATA SET 3
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
5
0
0
240
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(c)

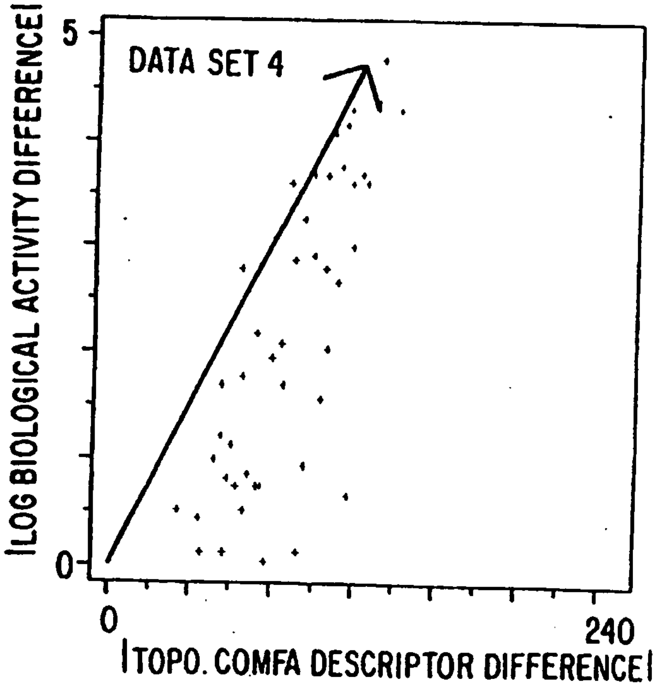

DATA SET 4
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
5
0
0
240
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG. 7(d)

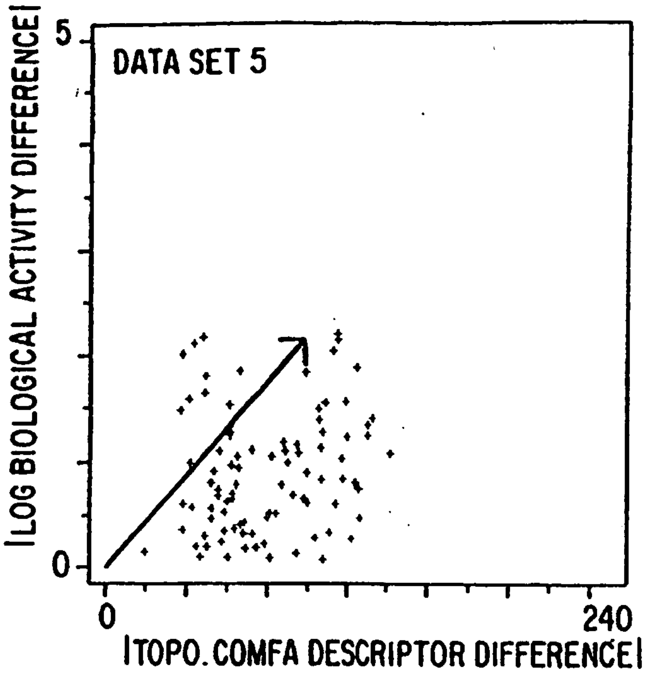

DATA SET 5
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(e)

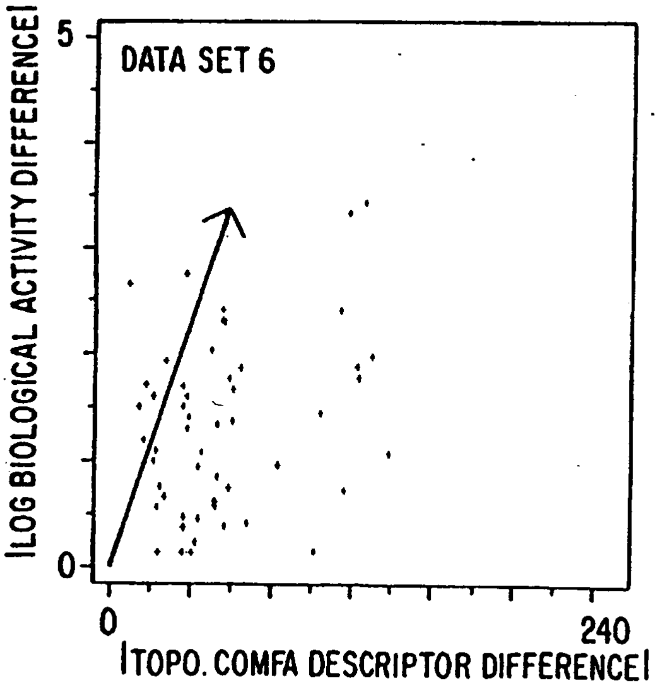

DATA SET 6
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(f)

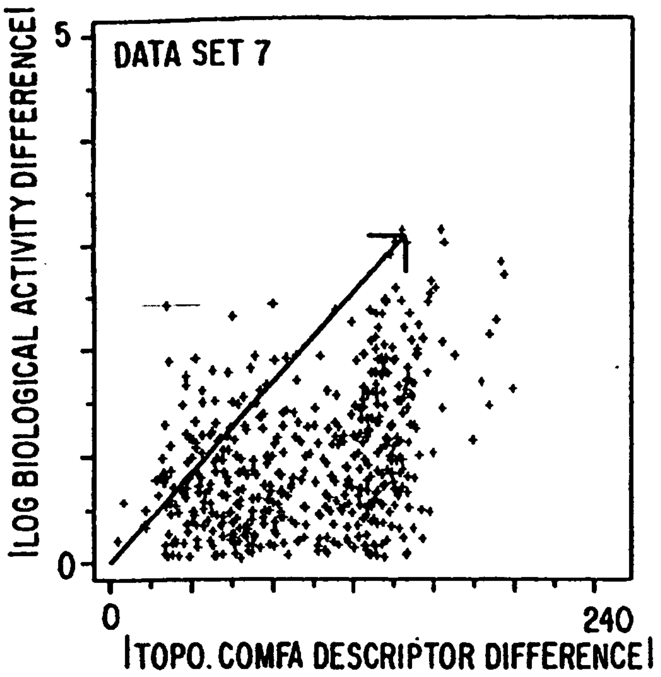

DATA SET 7
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(g)

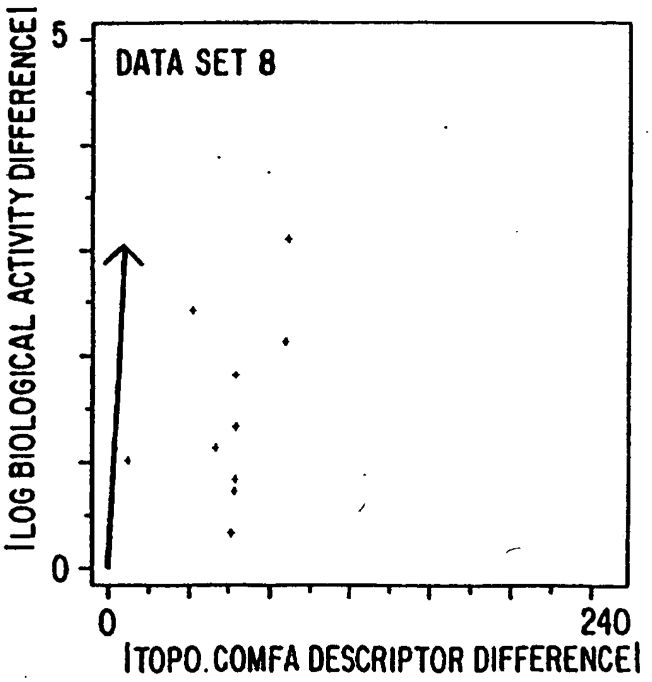

DATA SET 8
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(h)

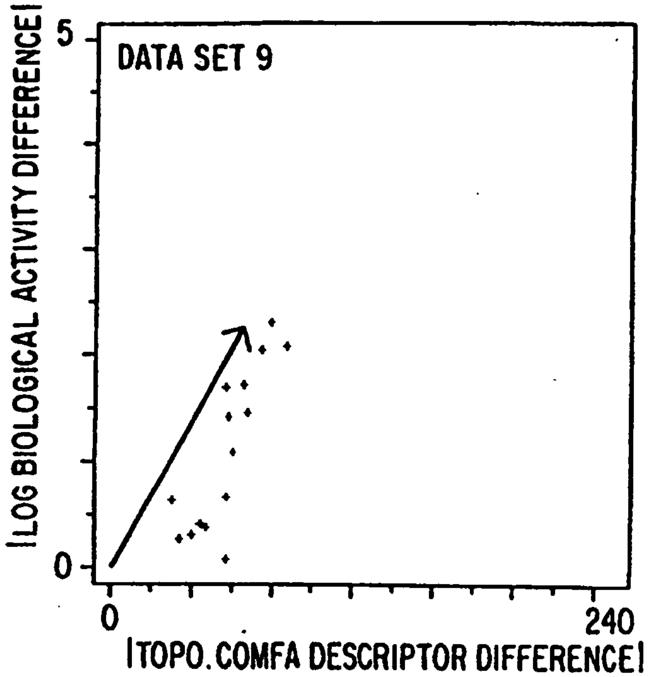

DATA SET 9
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO.COMFA DESCRIPTOR DIFFERENCE|

FIG.7(i)

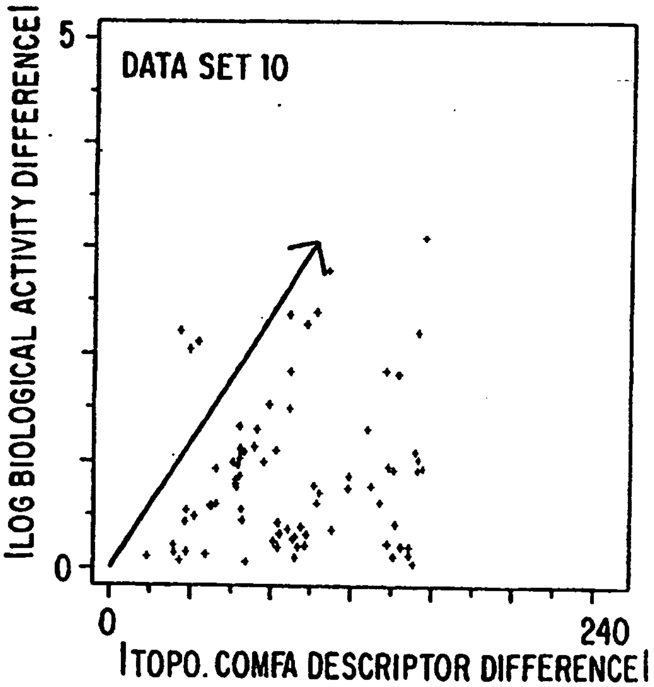

DATA SET 10
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO.COMFA DESCRIPTOR DIFFERENCE|

FIG.7(j)

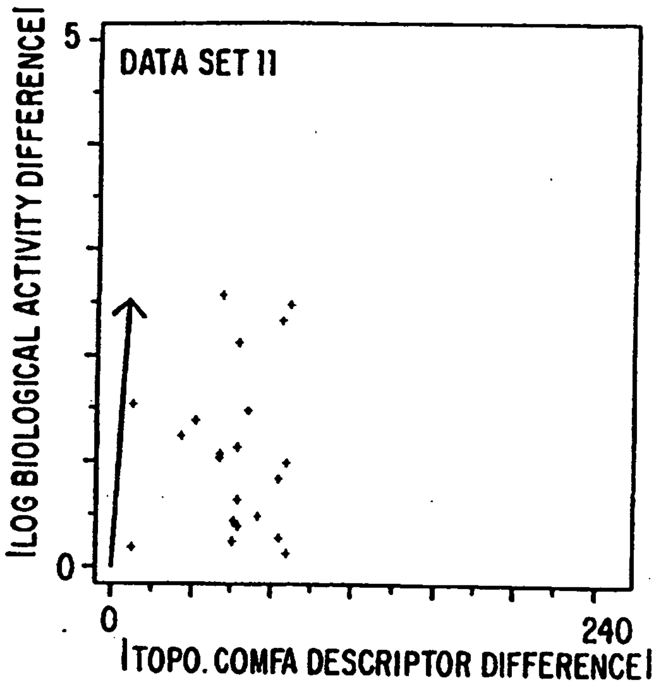

DATA SET 11
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
5
0
0
240
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(k)

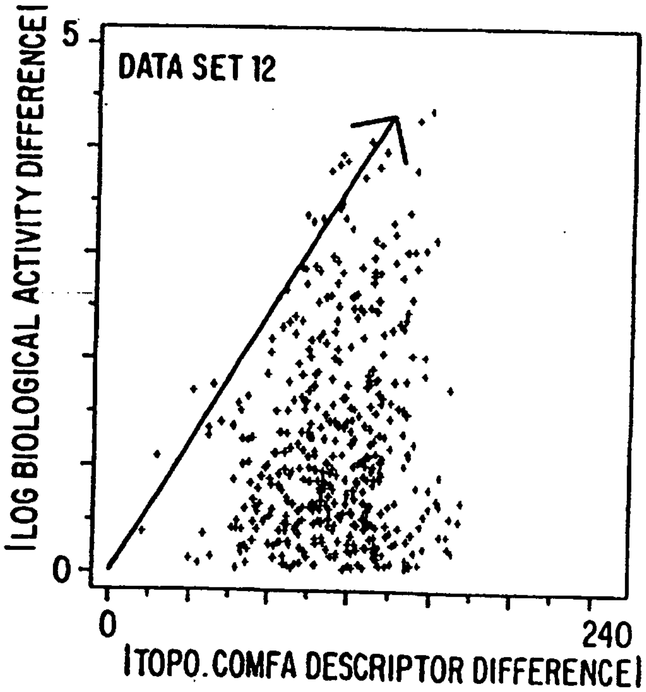

DATA SET 12
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
5
0
0
240
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG. 7(1)

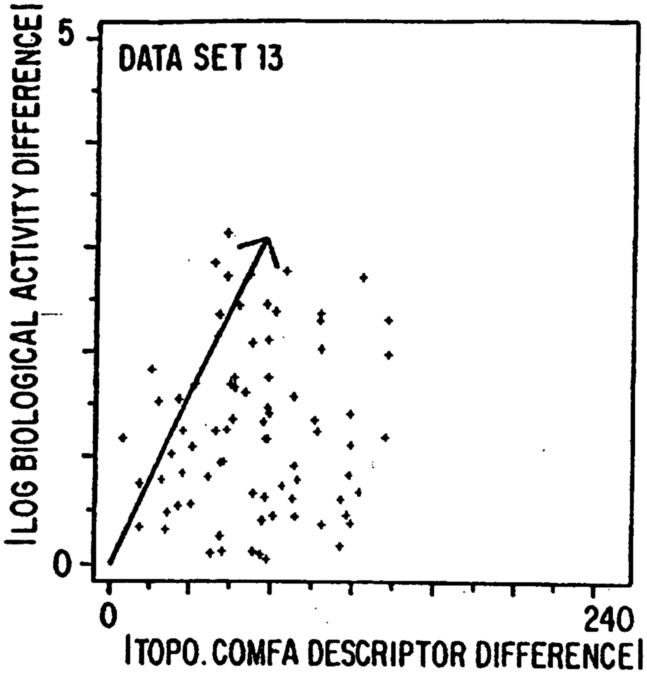

DATA SET 13
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
5
0
0
240
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG. 7(m)

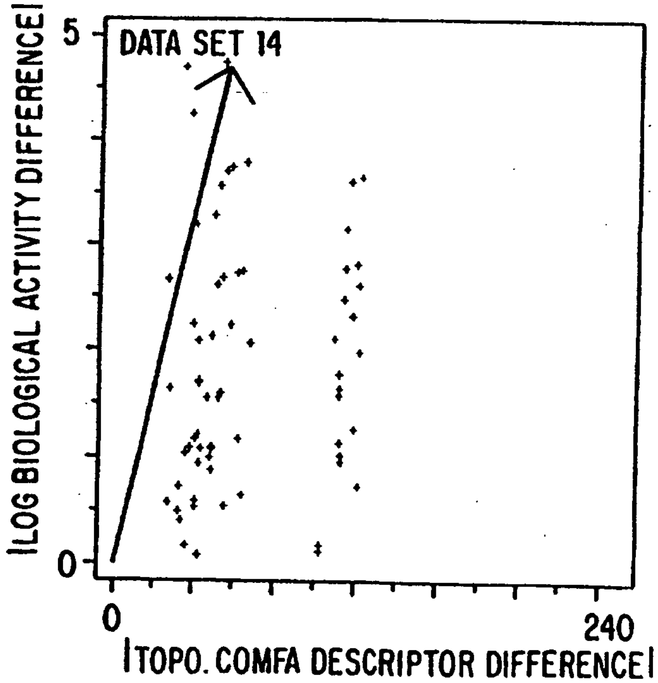

DATA SET 14
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
5
0
0
240
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG. 7(n)

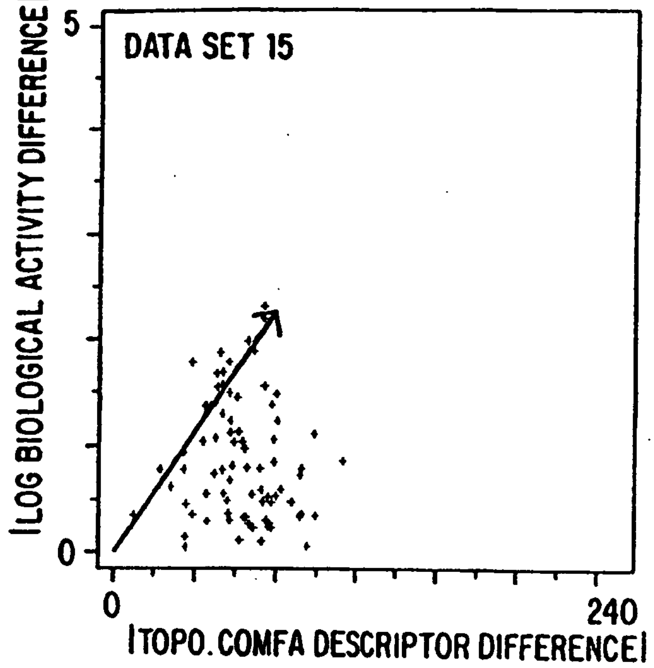

DATA SET 15
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(o)

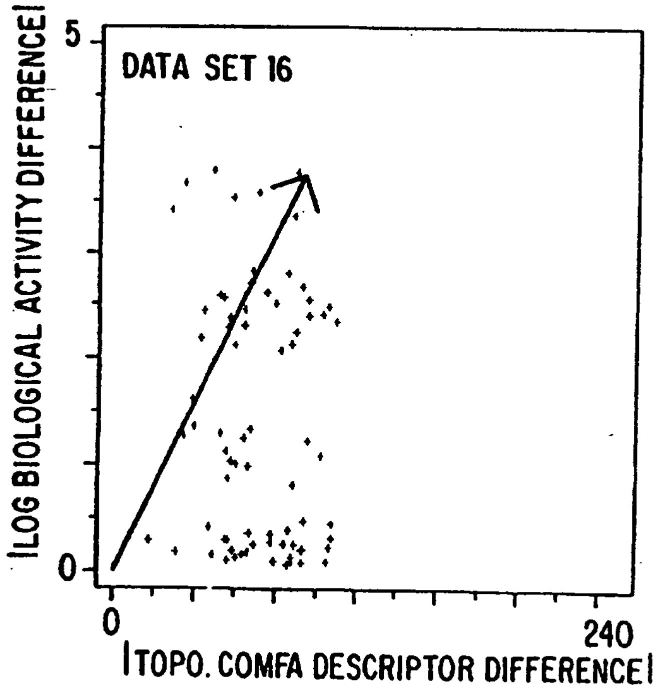

DATA SET 16
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(p)

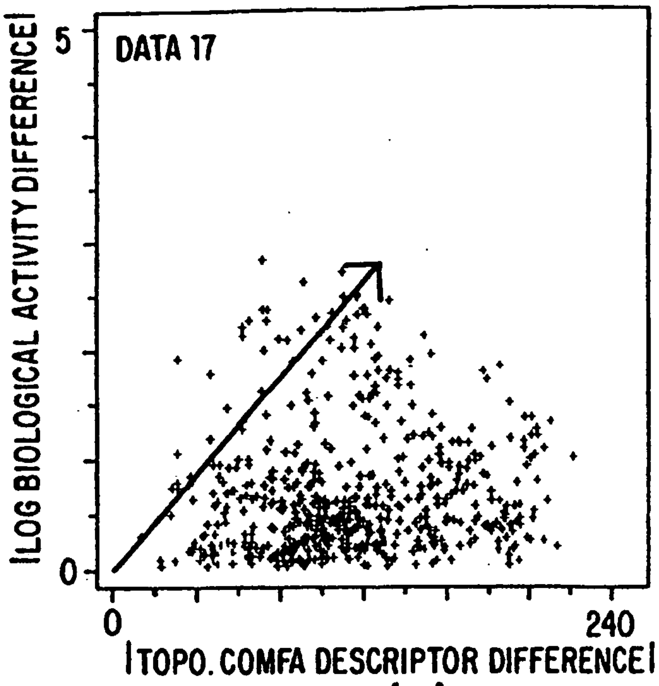

DATA 17
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(q)

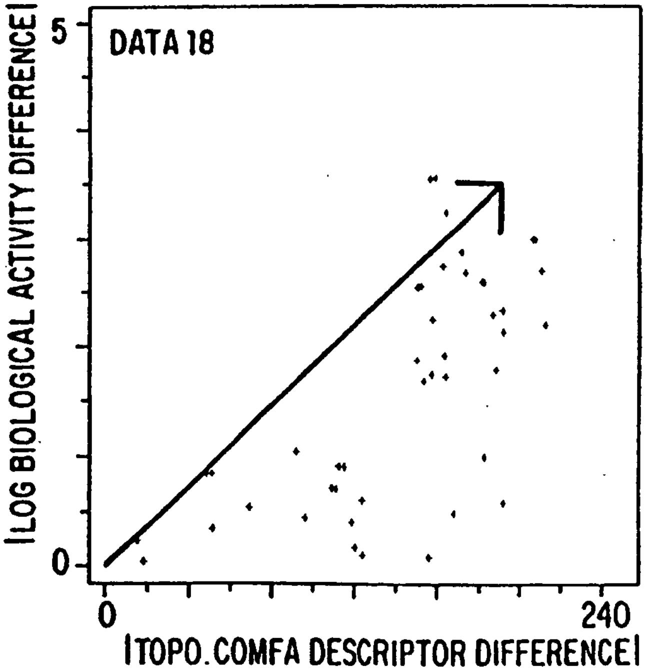

DATA 18
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG.7(r)

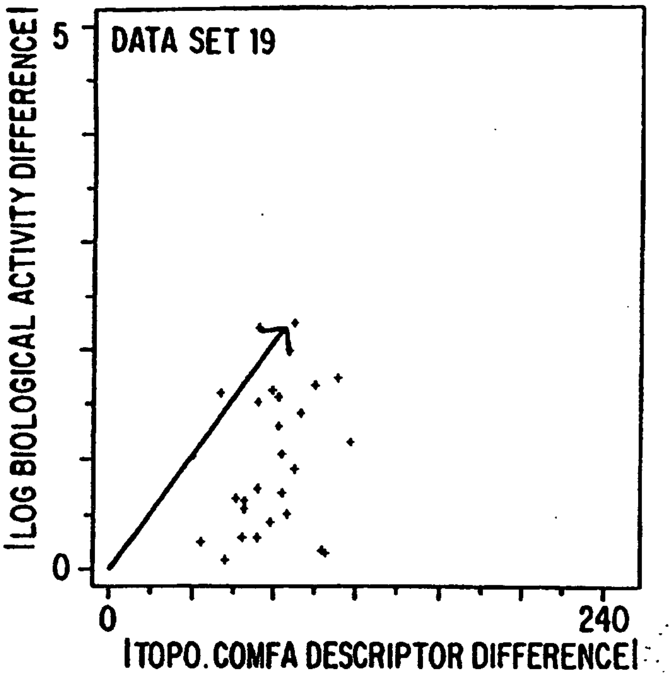

DATA SET 19
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG. 7(s)

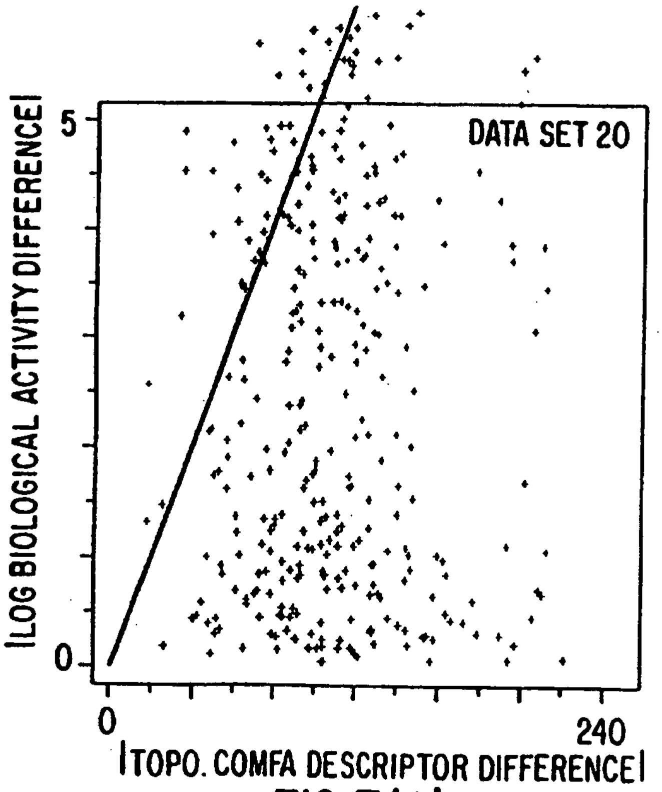

DATA SET 20
5
0
0
240
|LOG BIOLOGICAL ACTIVITY DIFFERENCE|
|TOPO. COMFA DESCRIPTOR DIFFERENCE|

FIG. 7(t)

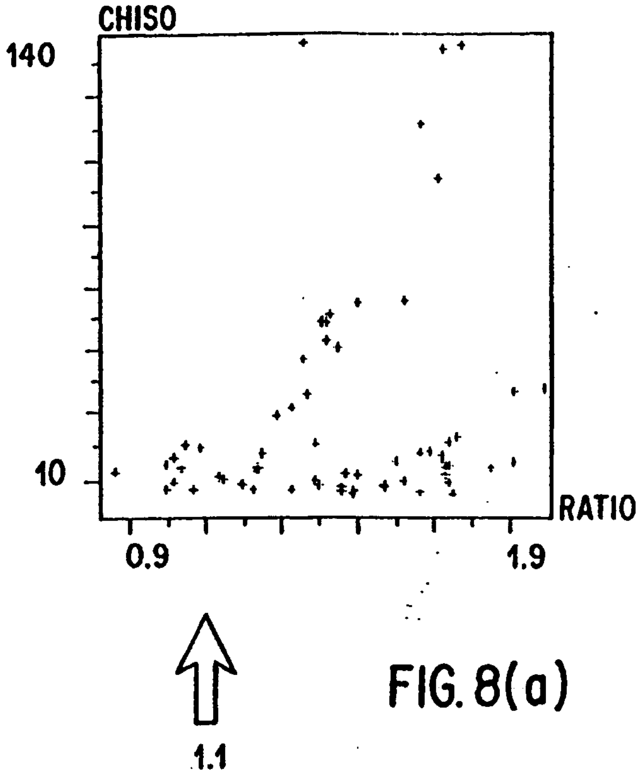


FIG. 8(a)

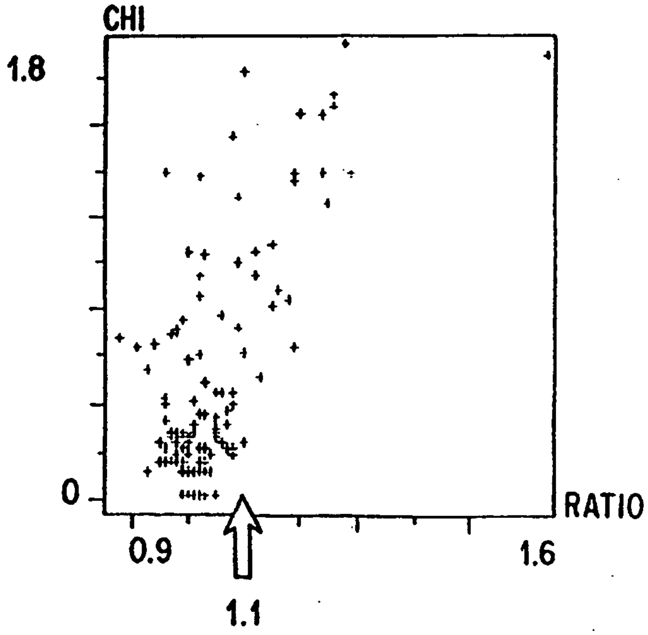


FIG. 8(b)

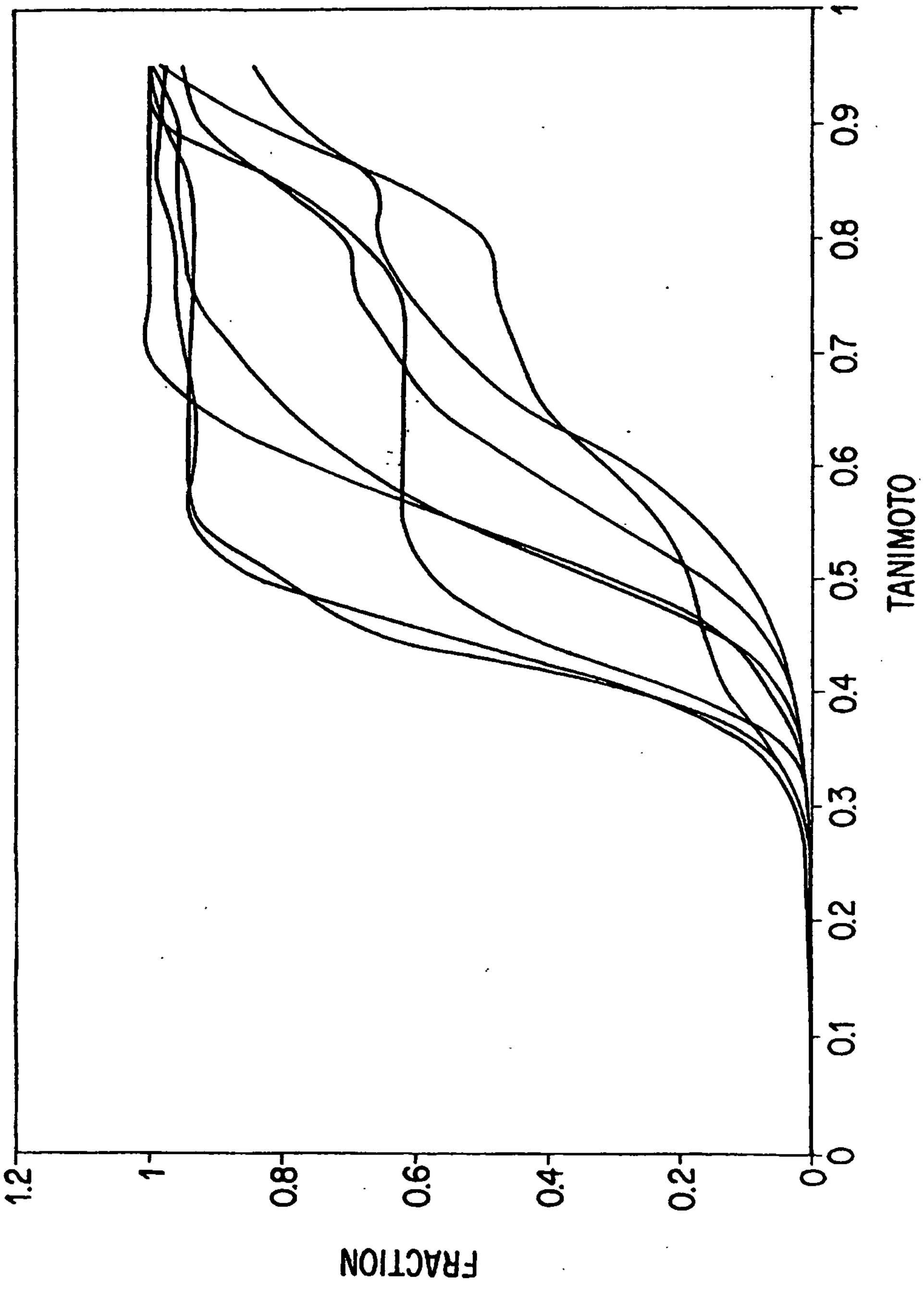


FIG. 9(a)

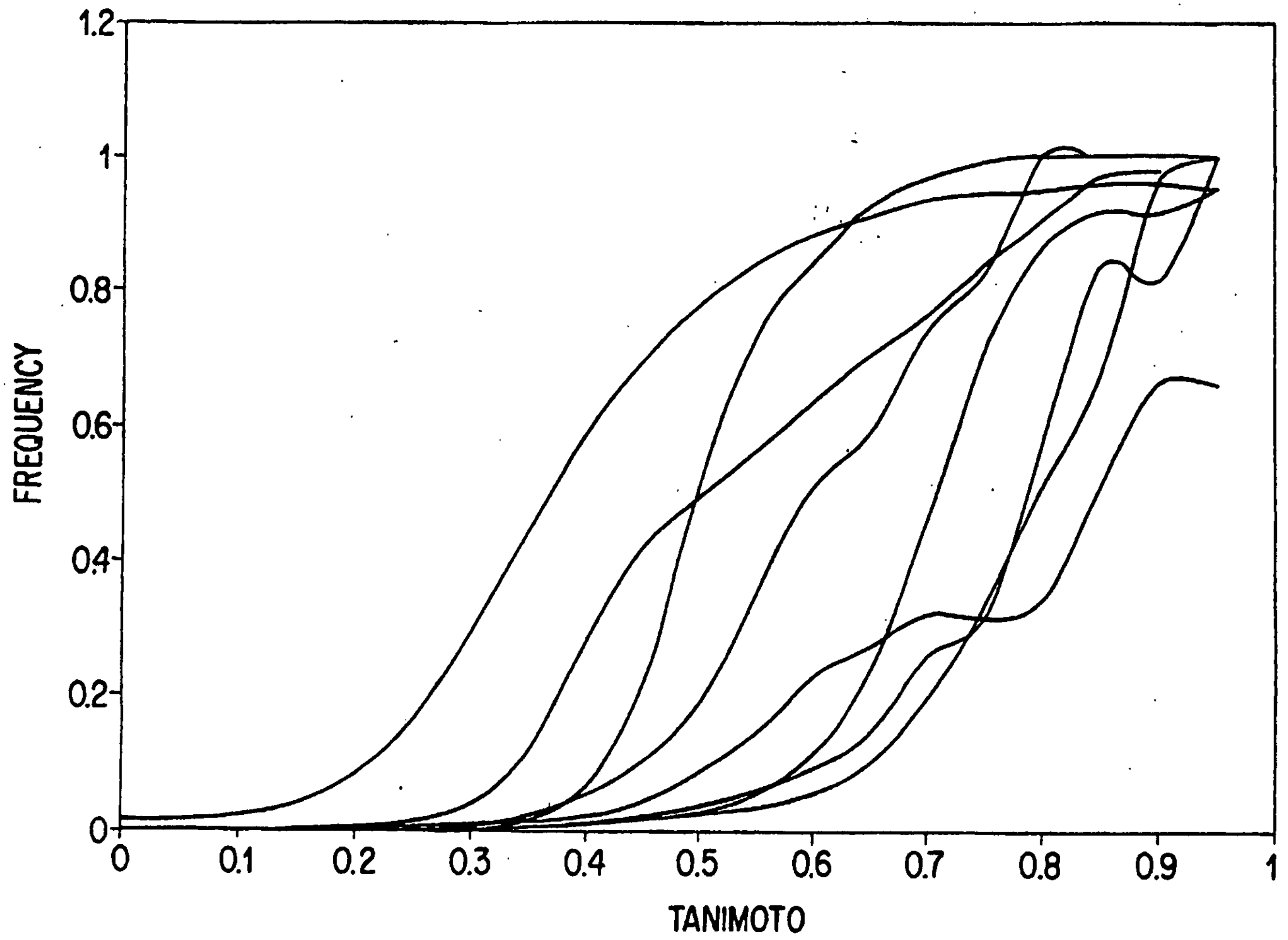


FIG. 9(b)

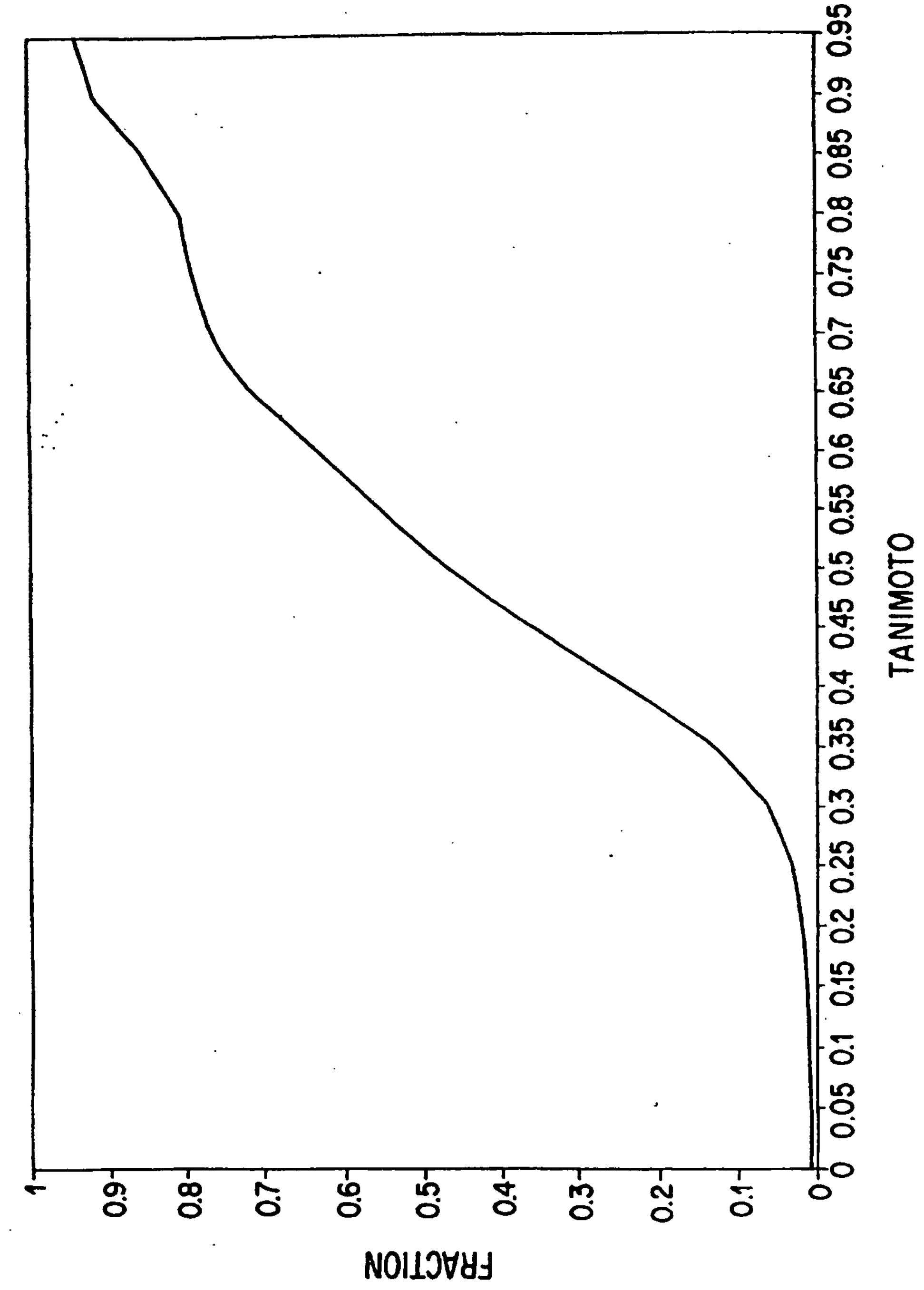

FRACTION
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
TANIMOTO
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95

FIG.9(c)

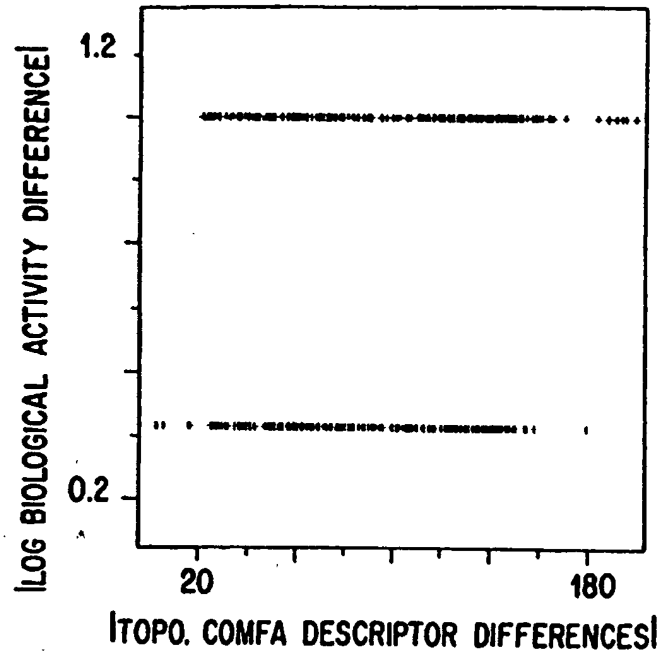


FIG.10(a)

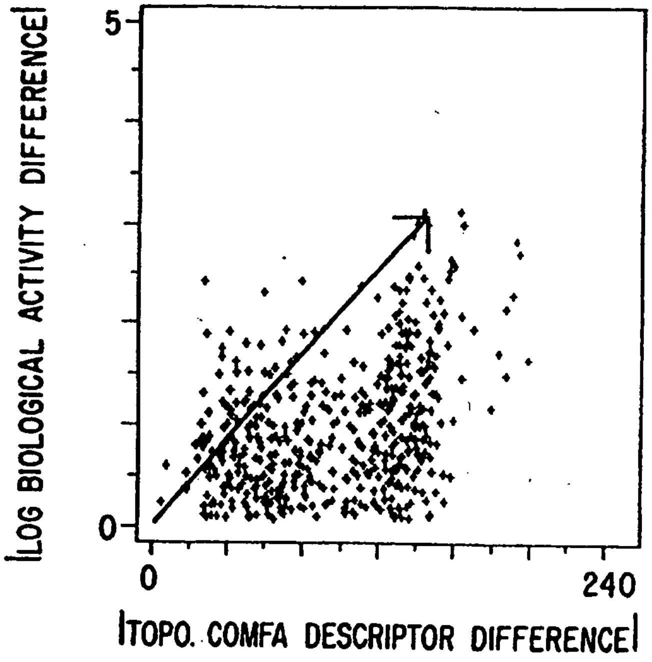
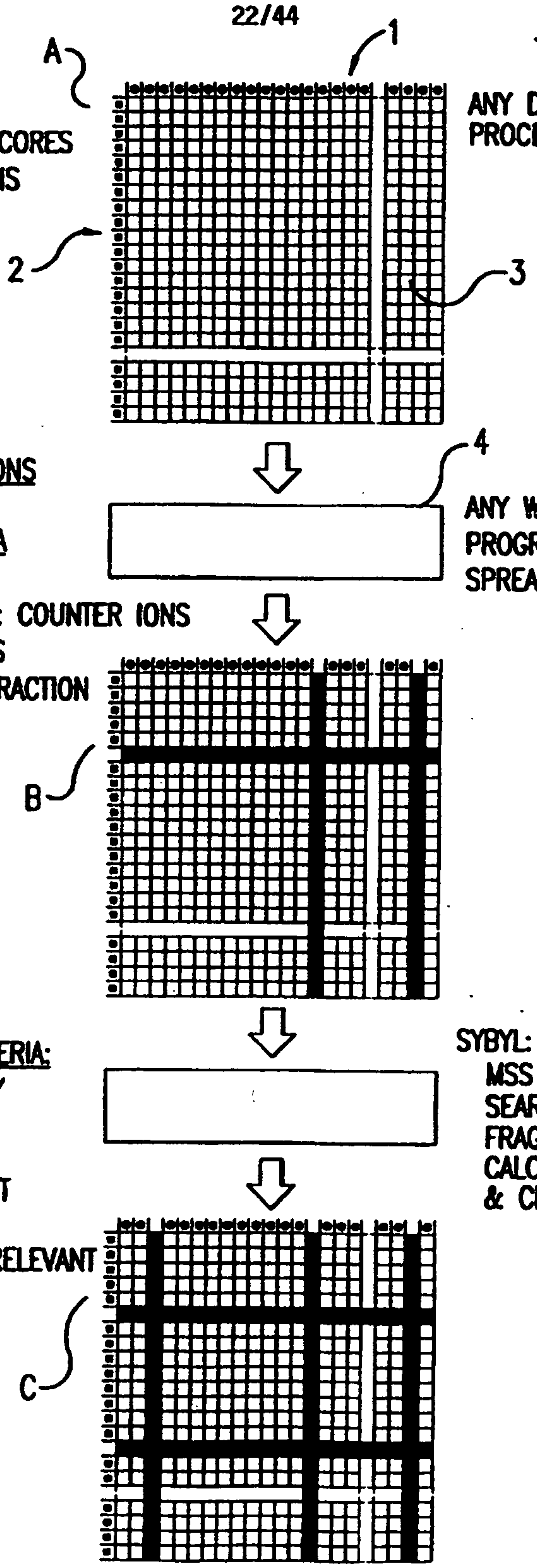


FIG.10(b)

FUNCTION

SOFTWARE



8.
SELECT CHEMISTRY
LIST REACTANTS & CORES
DEFINE COMBINATIONS

ANY DATABASE OR WORD PROCESSING PROGRAM

8.A
REMOVE REACTANTS:
NON-DIVERSITY REASONS

8A(i) GENERAL CRITERIA
METALS
TAUTOMERS/SALTS & COUNTER IONS
INTERFERING GROUPS
SYNTHESIS & EXTRACTION
PRICE/AVAILABILITY

ANY WORD PROCESSING PROGRAM OR MOLECULAR SPREAD SHEET (MSS)

8.(A)ii BIOLOGICAL CRITERIA:
TOXIC/METABOLICALLY HAZARDOUS
BIOAVAILABILITY
MOLECULAR WEIGHT
CLOGP
NON-BIOLOGICALLY RELEVANT GROUPS
SUGARS/MULTIPLE FUNCTIONALITIES

SYBYL: SPL SCRIPTS ON MSS INVOKING-UNITY SEARCHES FOR FRAGMENTS CALCULATIONS OF MW & CLOGP

FIG.11(a)

8B

REMOVE NON-DIVERSE REACTANTS:

3D STRUCTURE GENERATION

TOPOMERIC CONFORMER ALIGNMENT

CoMFA FIELD GENERATION:

HYDROGEN BOND FIELD GENERATION

ROTATABLE BOND FIELD ATTENUATION

CALCULATE FIELD DIFFERENCES

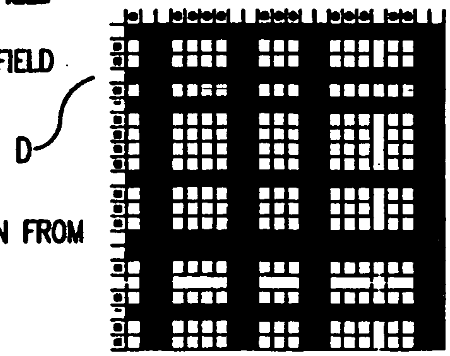
CLUSTER

REACTANT SELECTION FROM EACH CLUSTER

SYBYL: SELECTOR & CONCORD 3.2.1
SPECIALIZED SOFTWARE: APPENDIX "A"
(SYBYL: CoMFA STERIC & MSS)

SYBYL: HIERCHICAL CLUSTER – MSS

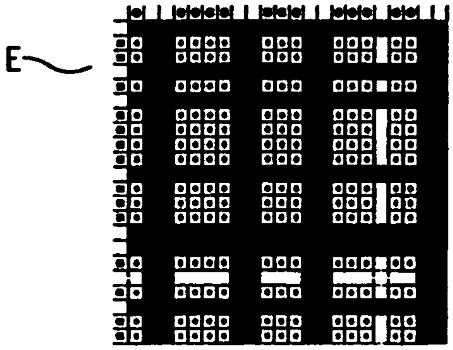
SYLBL: MSS



8C

COMBINE REACTANTS TO BUILD PRODUCTS

SYBYL: LEGION MSS



8D

REMOVE PRODUCTS:
NON-DIVERSITY REASONS

GENERAL CRITERIA:

SYBYL: SPL SCRIPTS ON MSS

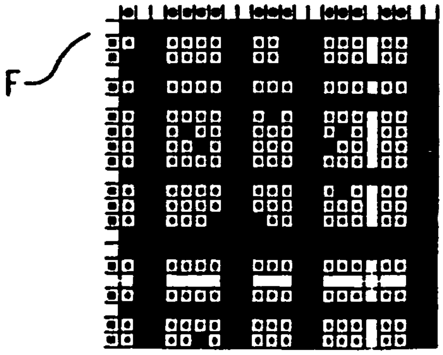


FIG.11(b)

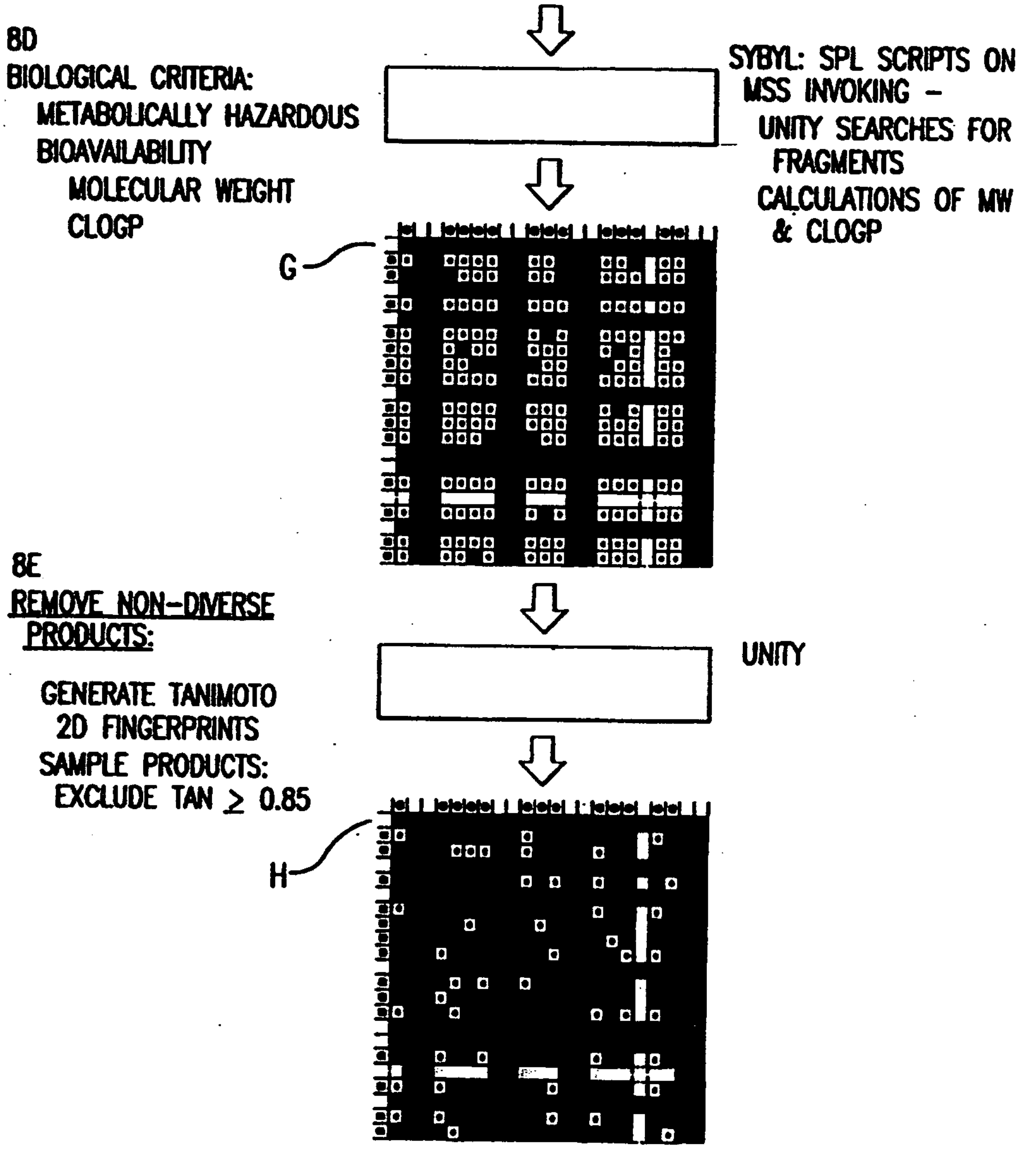

8D
BIOLOGICAL CRITERIA:
METABOLICALLY HAZARDOUS
BIOAVAILABILITY
MOLECULAR WEIGHT
CLOGP
SYBYL: SPL SCRIPTS ON MSS INVOKING -
UNITY SEARCHES FOR FRAGMENTS
CALCULATIONS OF MW & CLOGP
G
8E
REMOVE NON-DIVERSE PRODUCTS:
GENERATE TANIMOTO 2D FINGERPRINTS
SAMPLE PRODUCTS:
EXCLUDE TAN ≥ 0.85
UNITY
H

FIG.11(c)

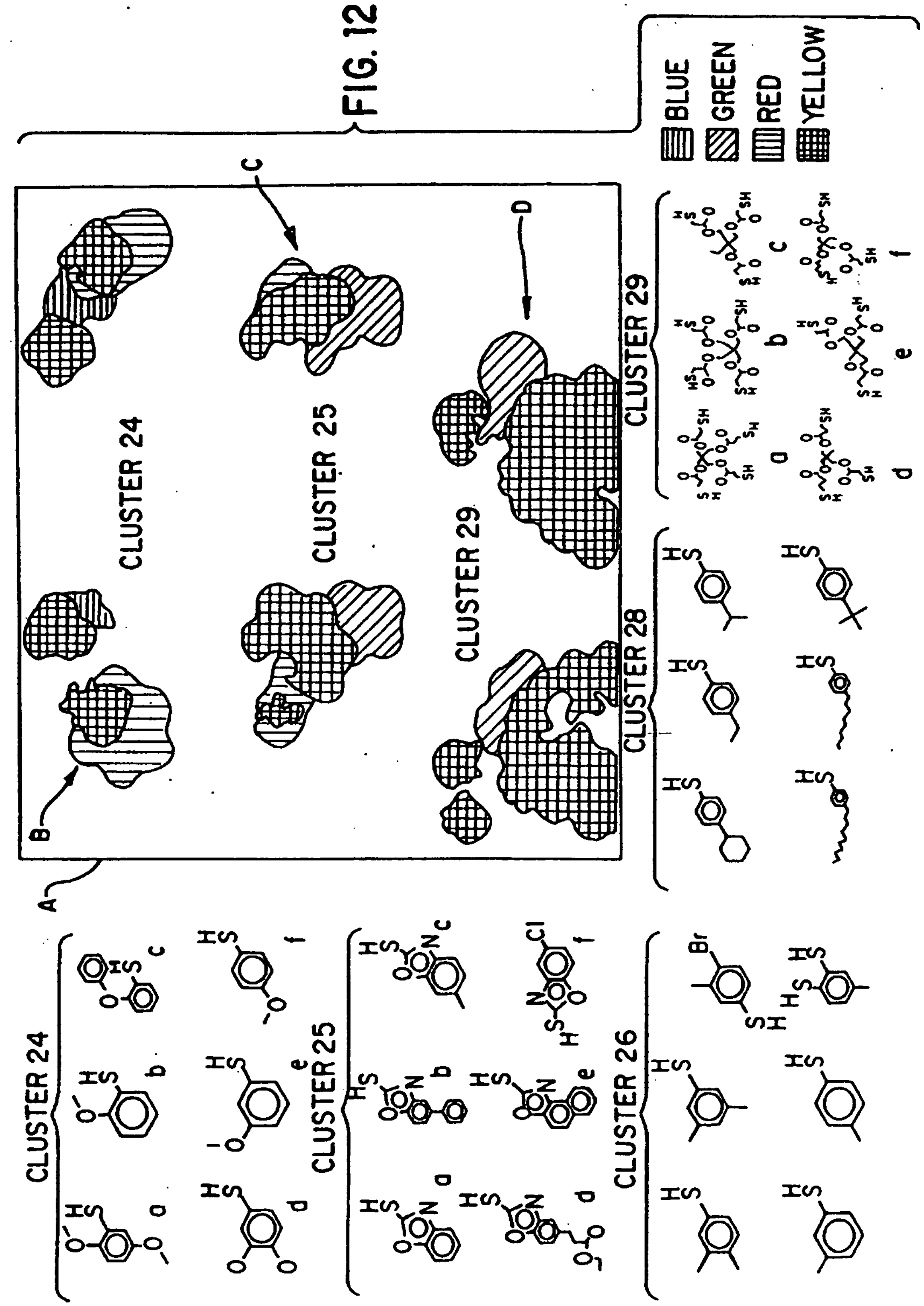

A
B
C
D
CLUSTER 24
CLUSTER 25
CLUSTER 29
CLUSTER 24
a
b
c
d
e
f
CLUSTER 25
a
b
c
d
e
f
CLUSTER 26
CLUSTER 28
CLUSTER 29
a
b
c
d
e
f
▤ BLUE
▨ GREEN
▥ RED
▦ YELLOW

FIG. 12

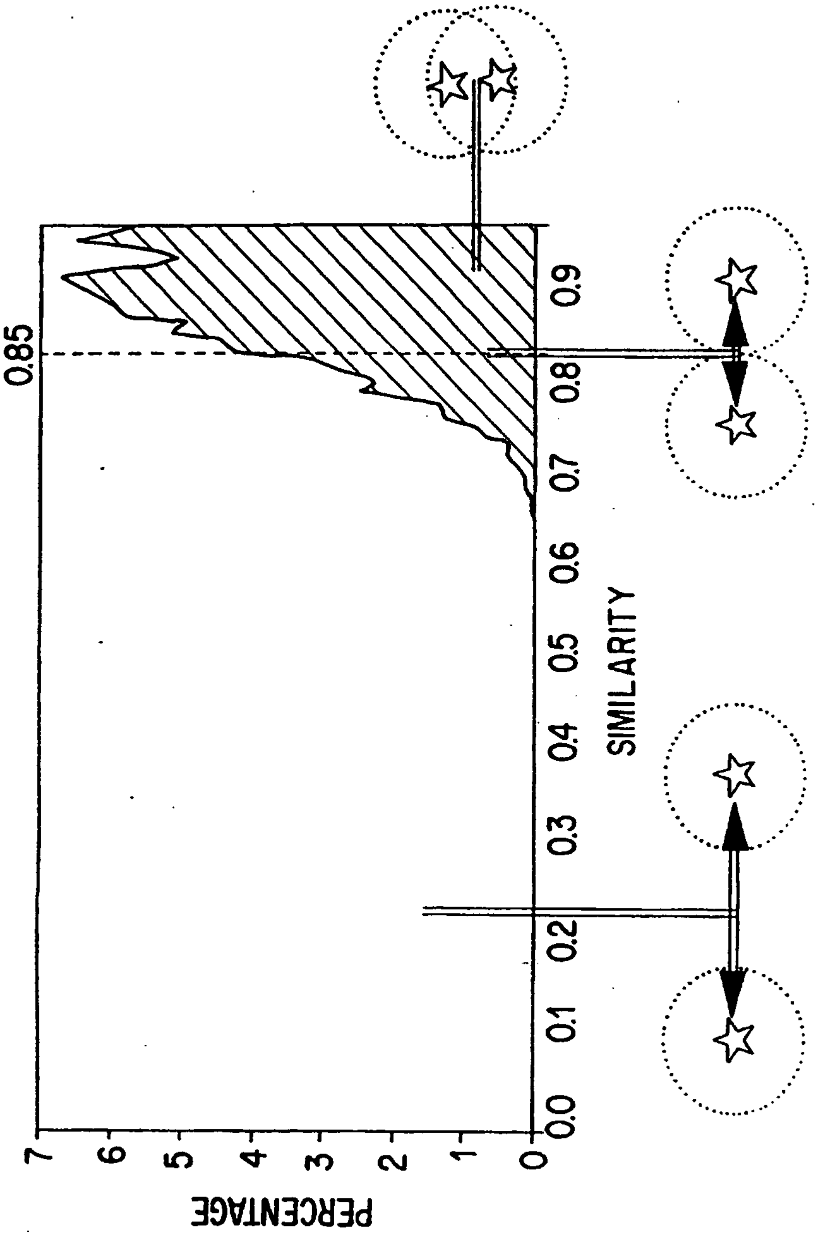

0.85
PERCENTAGE
7
6
5
4
3
2
1
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
SIMILARITY

FIG.13

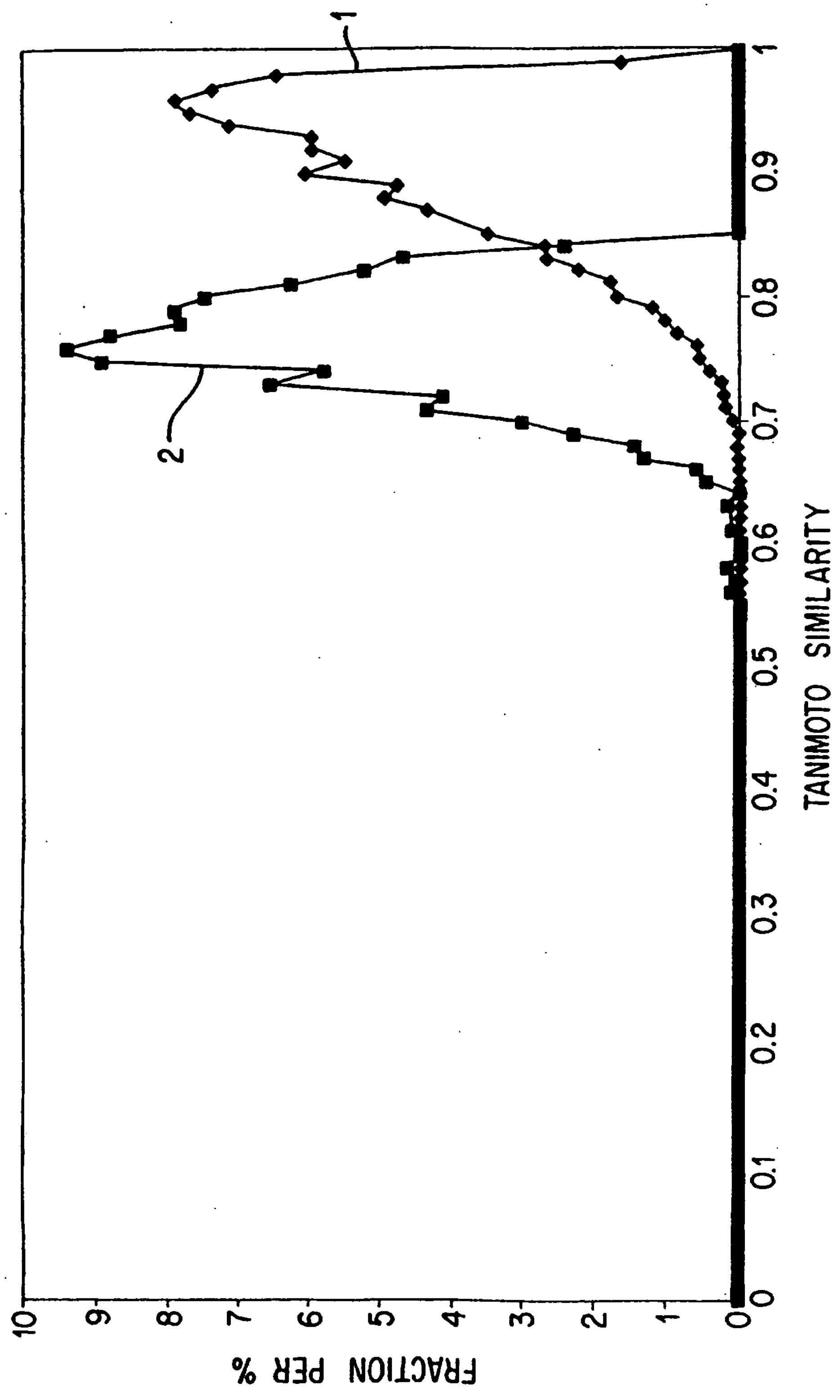

FRACTION PER %
10
9
8
7
6
5
4
3
2
1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
TANIMOTO SIMILARITY
1
2

FIG.14

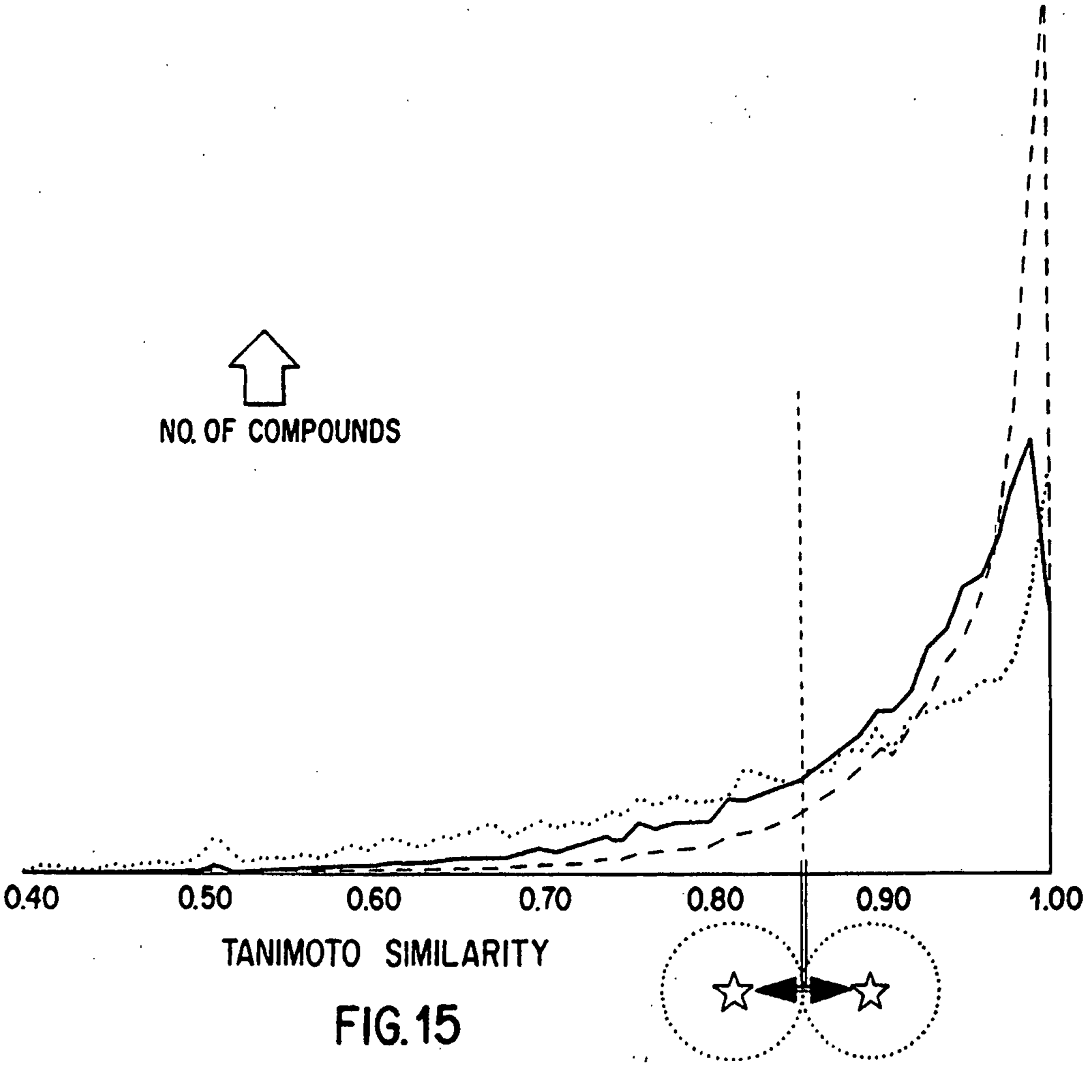

NO. OF COMPOUNDS
0.40
0.50
0.60
0.70
0.80
0.90
1.00
TANIMOTO SIMILARITY

FIG.15

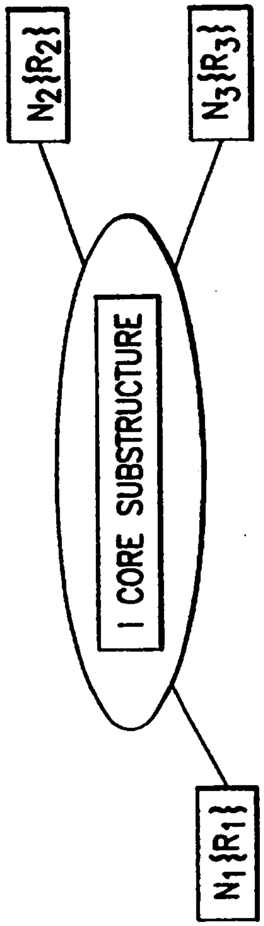


FIG.16

FIGURE 17

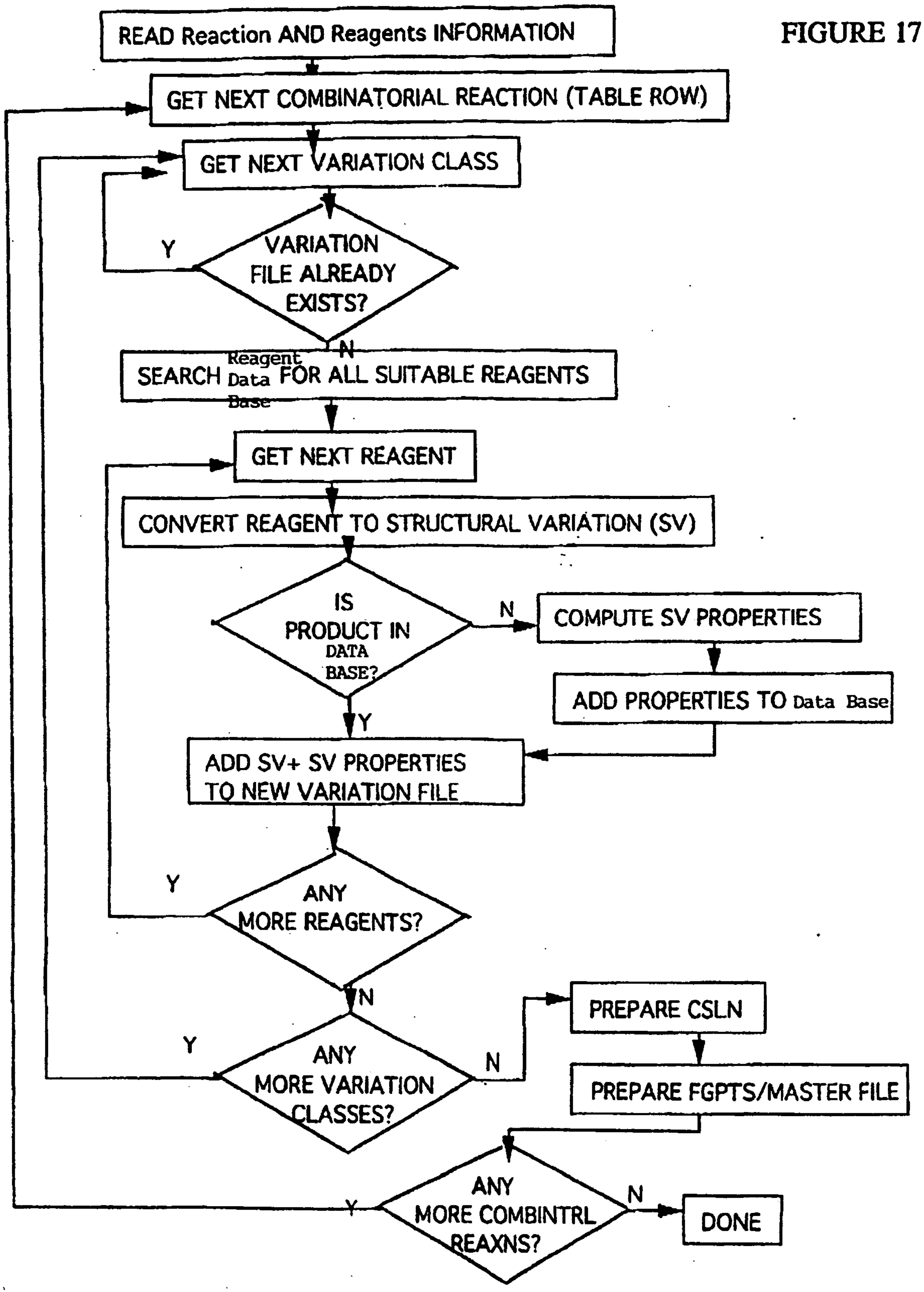

READ Reaction AND Reagents INFORMATION
GET NEXT COMBINATORIAL REACTION (TABLE ROW)
GET NEXT VARIATION CLASS
VARIATION FILE ALREADY EXISTS?
Y
N
SEARCH Reagent Data Base FOR ALL SUITABLE REAGENTS
GET NEXT REAGENT
CONVERT REAGENT TO STRUCTURAL VARIATION (SV)
IS PRODUCT IN DATA BASE?
N
COMPUTE SV PROPERTIES
ADD PROPERTIES TO Data Base
Y
ADD SV+ SV PROPERTIES TO NEW VARIATION FILE
Y
ANY MORE REAGENTS?
N
Y
ANY MORE VARIATION CLASSES?
N
PREPARE CSLN
PREPARE FGPTS/MASTER FILE
Y
ANY MORE COMBINTRL REAXNS?
N
DONE

FIGURE 18

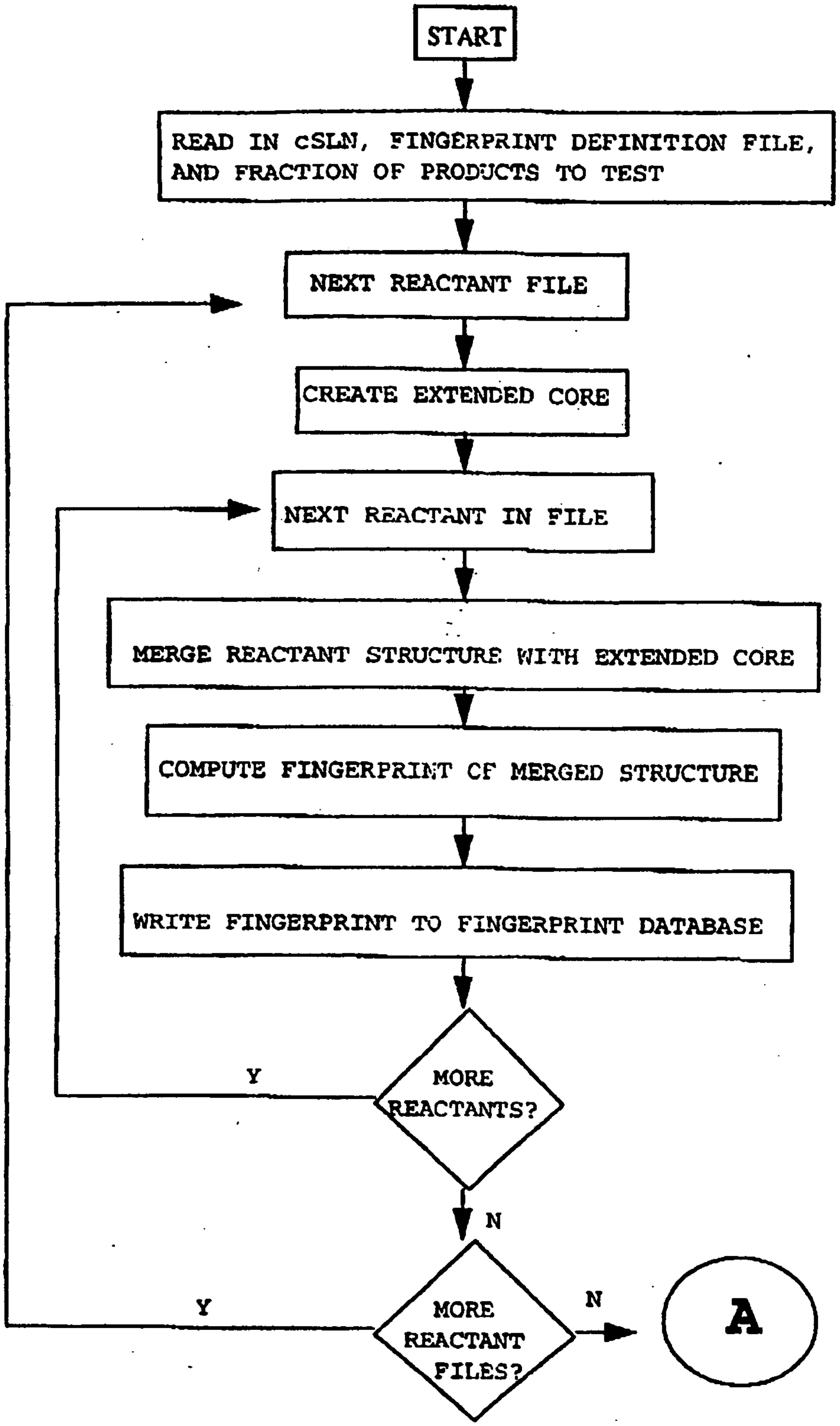

START
READ IN cSLN, FINGERPRINT DEFINITION FILE, AND FRACTION OF PRODUCTS TO TEST
NEXT REACTANT FILE
CREATE EXTENDED CORE
NEXT REACTANT IN FILE
MERGE REACTANT STRUCTURE WITH EXTENDED CORE
COMPUTE FINGERPRINT OF MERGED STRUCTURE
WRITE FINGERPRINT TO FINGERPRINT DATABASE
MORE REACTANTS?
Y
N
MORE REACTANT FILES?
Y
N
A

FIGURE 19

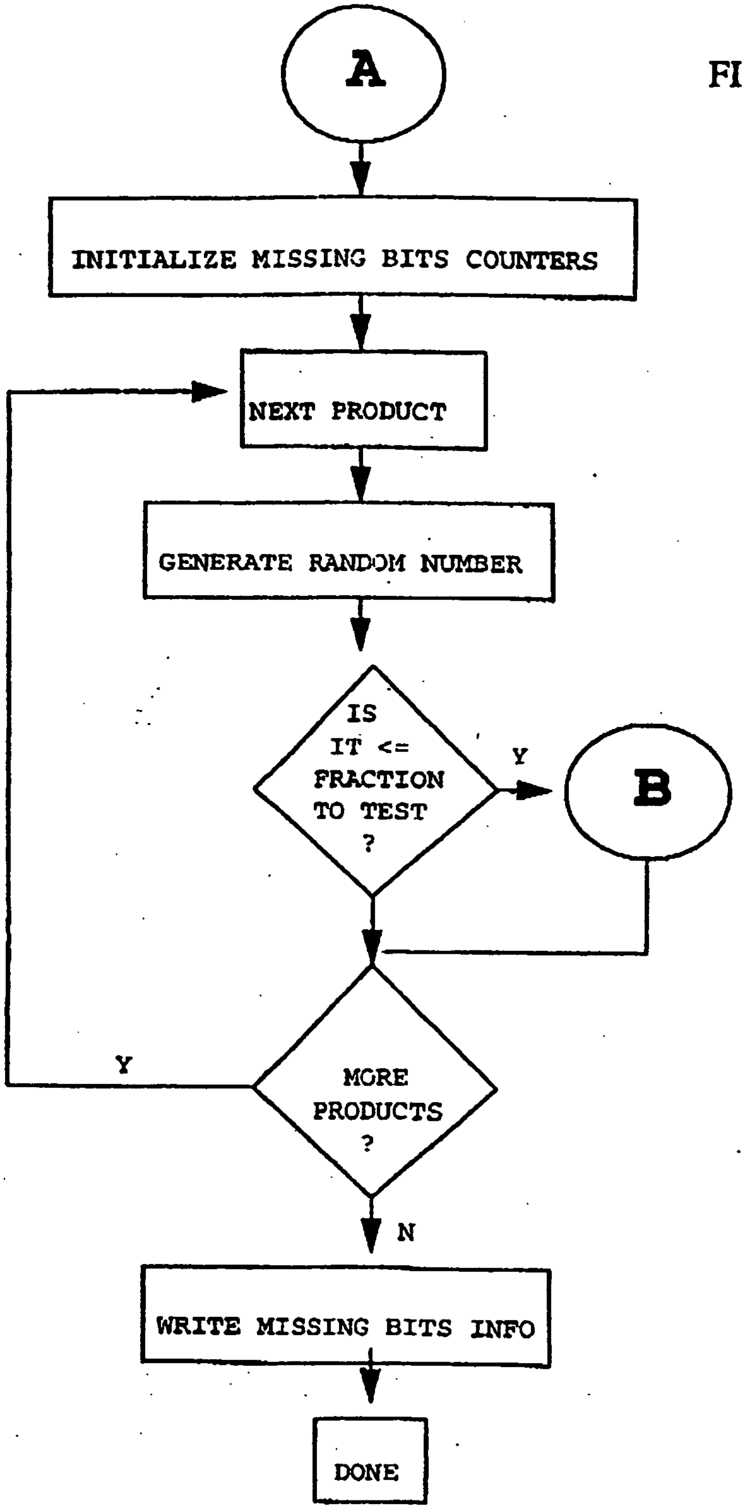

A
INITIALIZE MISSING BITS COUNTERS
NEXT PRODUCT
GENERATE RANDOM NUMBER
IS IT <= FRACTION TO TEST ?
Y
B
MORE PRODUCTS ?
Y
N
WRITE MISSING BITS INFO
DONE

FIGURE 20

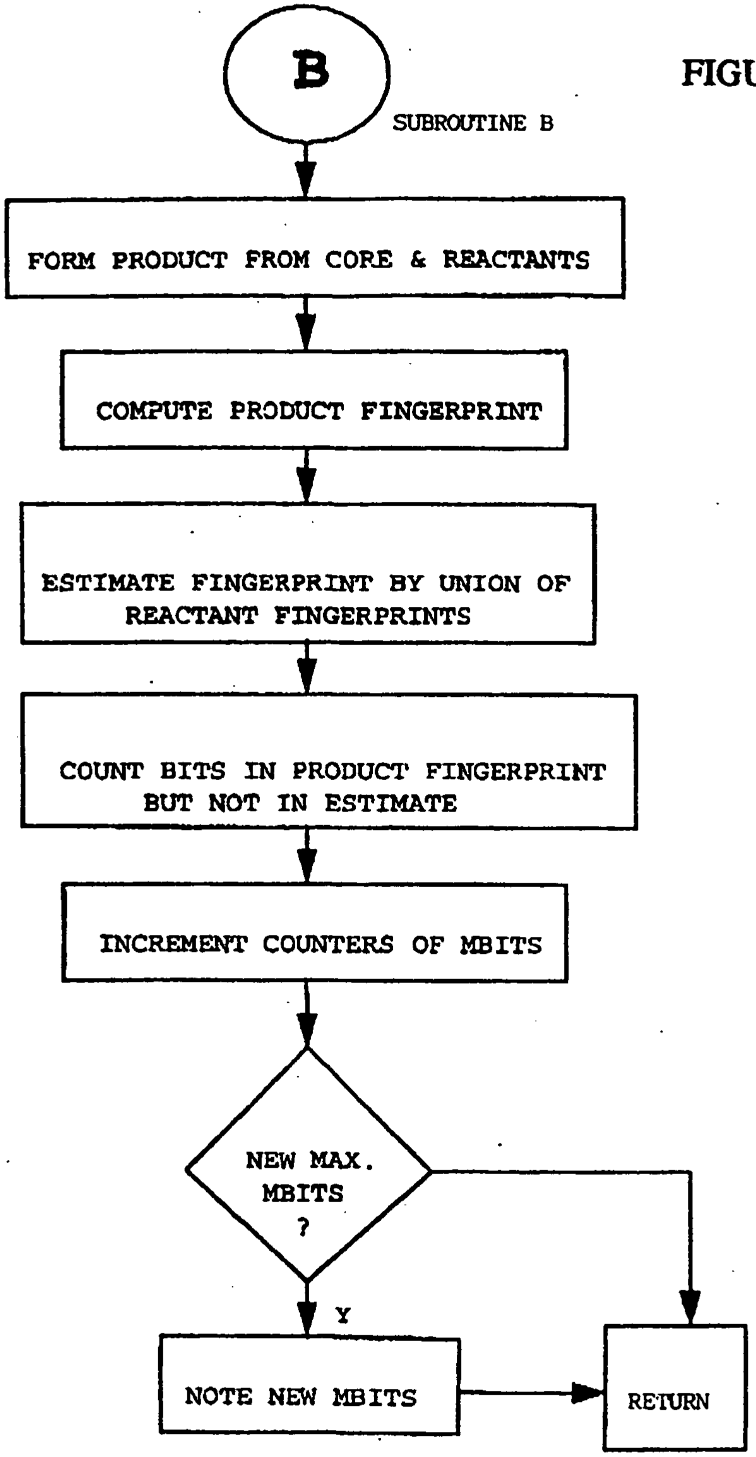

B
SUBROUTINE B
FORM PRODUCT FROM CORE & REACTANTS
COMPUTE PRODUCT FINGERPRINT
ESTIMATE FINGERPRINT BY UNION OF REACTANT FINGERPRINTS
COUNT BITS IN PRODUCT FINGERPRINT BUT NOT IN ESTIMATE
INCREMENT COUNTERS OF MBITS
NEW MAX. MBITS ?
Y
NOTE NEW MBITS
RETURN

FIGURE 21

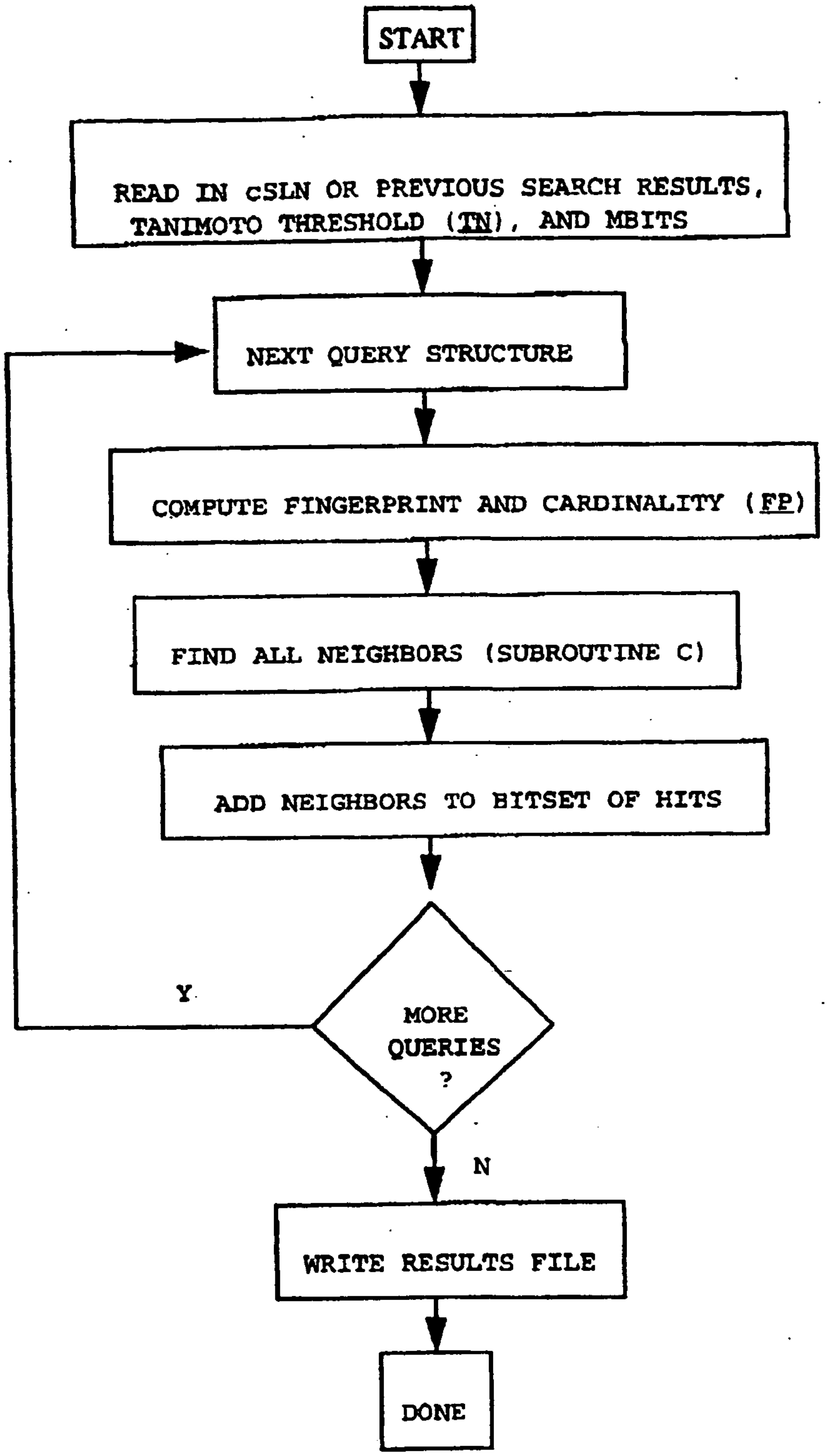

START
READ IN cSLN OR PREVIOUS SEARCH RESULTS, TANIMOTO THRESHOLD (TN), AND MBITS
NEXT QUERY STRUCTURE
COMPUTE FINGERPRINT AND CARDINALITY (FP)
FIND ALL NEIGHBORS (SUBROUTINE C)
ADD NEIGHBORS TO BITSET OF HITS
MORE QUERIES ?
Y
N
WRITE RESULTS FILE
DONE

FIGURE 22

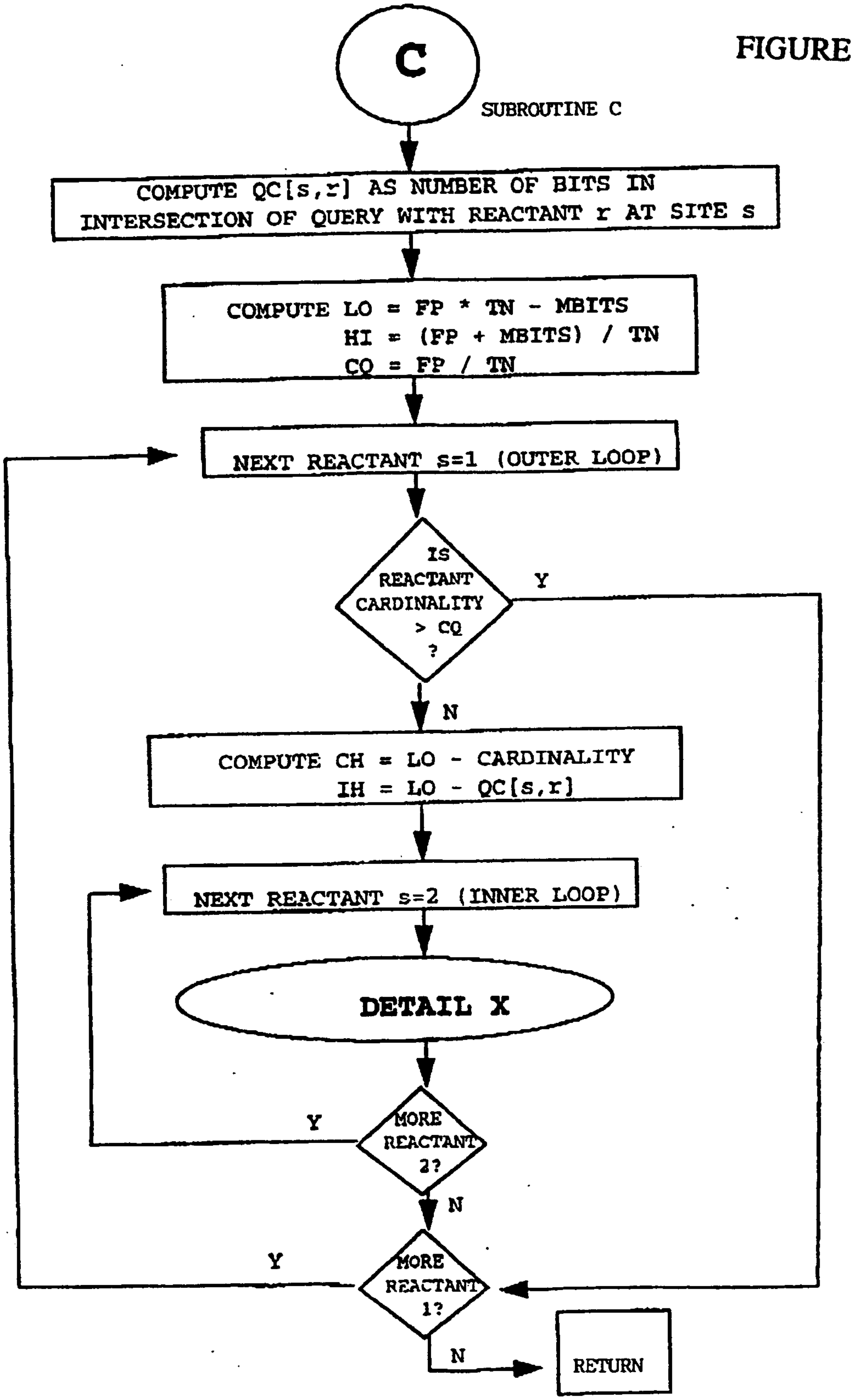

C
SUBROUTINE C
COMPUTE QC[s,r] AS NUMBER OF BITS IN INTERSECTION OF QUERY WITH REACTANT r AT SITE s
COMPUTE LO = FP * TN - MBITS
HI = (FP + MBITS) / TN
CQ = FP / TN
NEXT REACTANT s=1 (OUTER LOOP)
IS REACTANT CARDINALITY > CQ ?
Y
N
COMPUTE CH = LO - CARDINALITY
IH = LO - QC[s,r]
NEXT REACTANT s=2 (INNER LOOP)
DETAIL X
Y
MORE REACTANT 2?
N
Y
MORE REACTANT 1?
N
RETURN

FIGURE 23

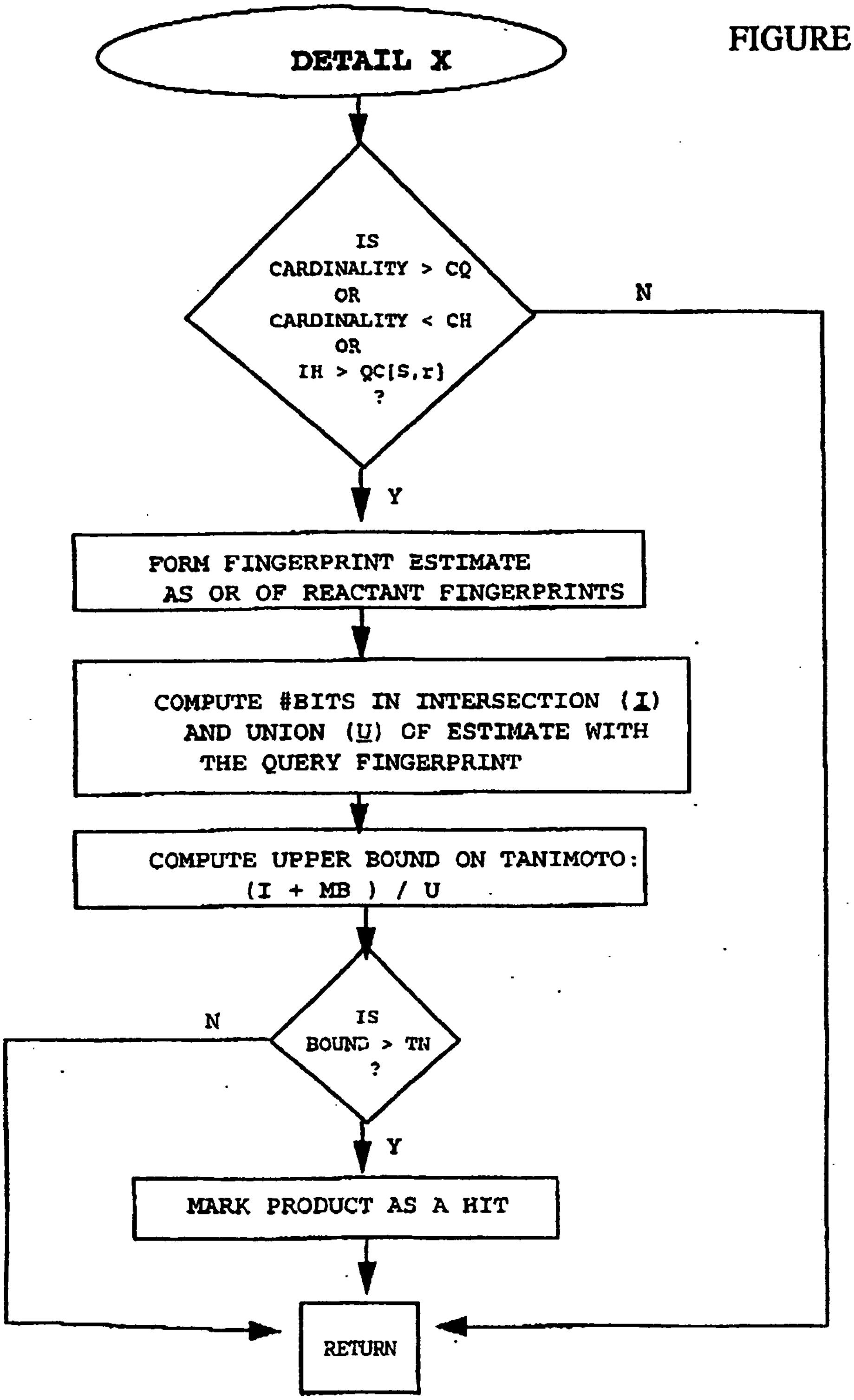

DETAIL X
IS CARDINALITY > CQ OR CARDINALITY < CH OR IH > QC(S,r) ?
N
Y
FORM FINGERPRINT ESTIMATE AS OR OF REACTANT FINGERPRINTS
COMPUTE #BITS IN INTERSECTION (I) AND UNION (U) OF ESTIMATE WITH THE QUERY FINGERPRINT
COMPUTE UPPER BOUND ON TANIMOTO: (I + MB) / U
IS BOUND > TN ?
N
Y
MARK PRODUCT AS A HIT
RETURN

FIGURE 24

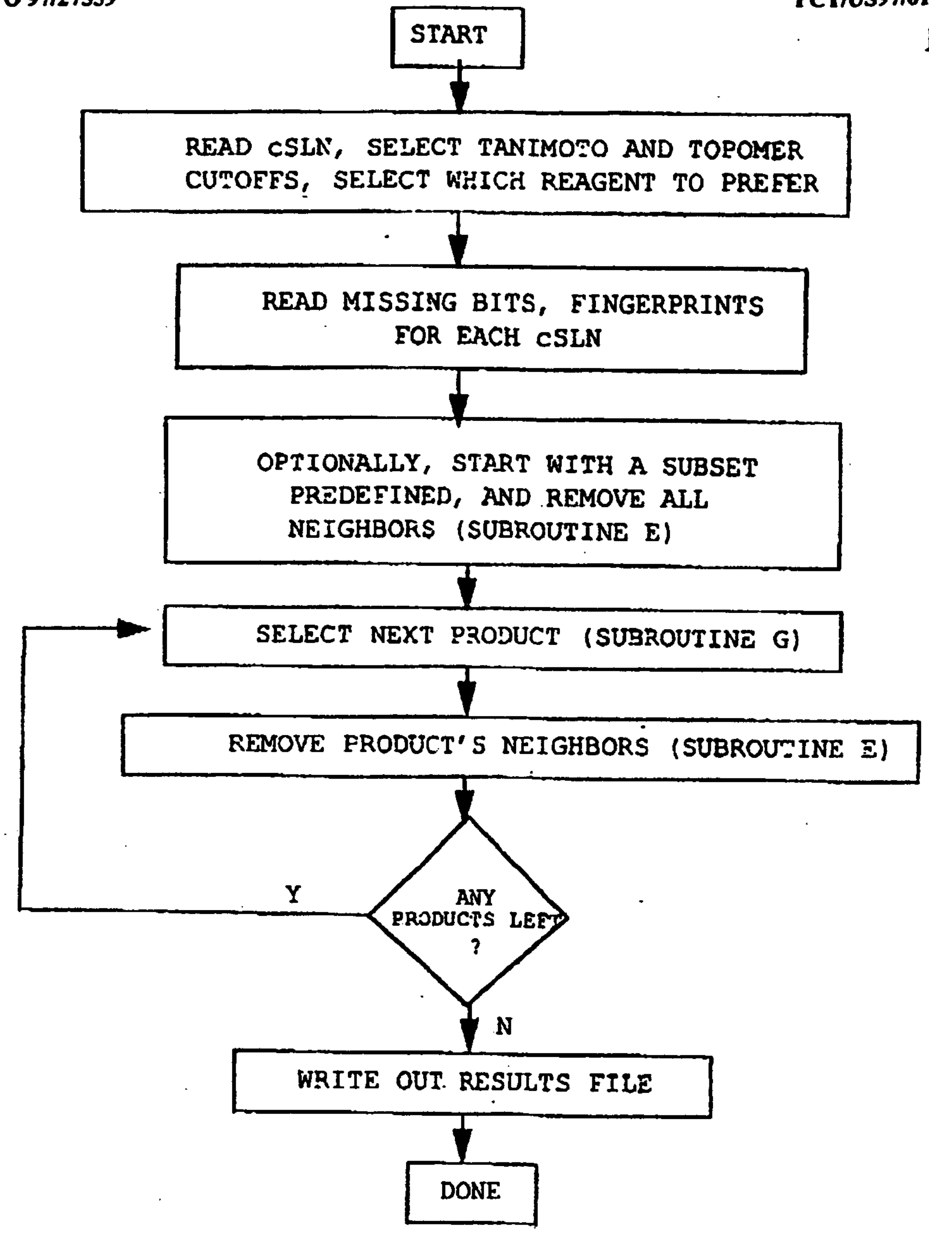

START
READ cSLN, SELECT TANIMOTO AND TOPOMER CUTOFFS, SELECT WHICH REAGENT TO PREFER
READ MISSING BITS, FINGERPRINTS FOR EACH cSLN
OPTIONALLY, START WITH A SUBSET PREDEFINED, AND REMOVE ALL NEIGHBORS (SUBROUTINE E)
SELECT NEXT PRODUCT (SUBROUTINE G)
REMOVE PRODUCT'S NEIGHBORS (SUBROUTINE E)
ANY PRODUCTS LEFT ?
Y
N
WRITE OUT RESULTS FILE
DONE

FIGURE 25

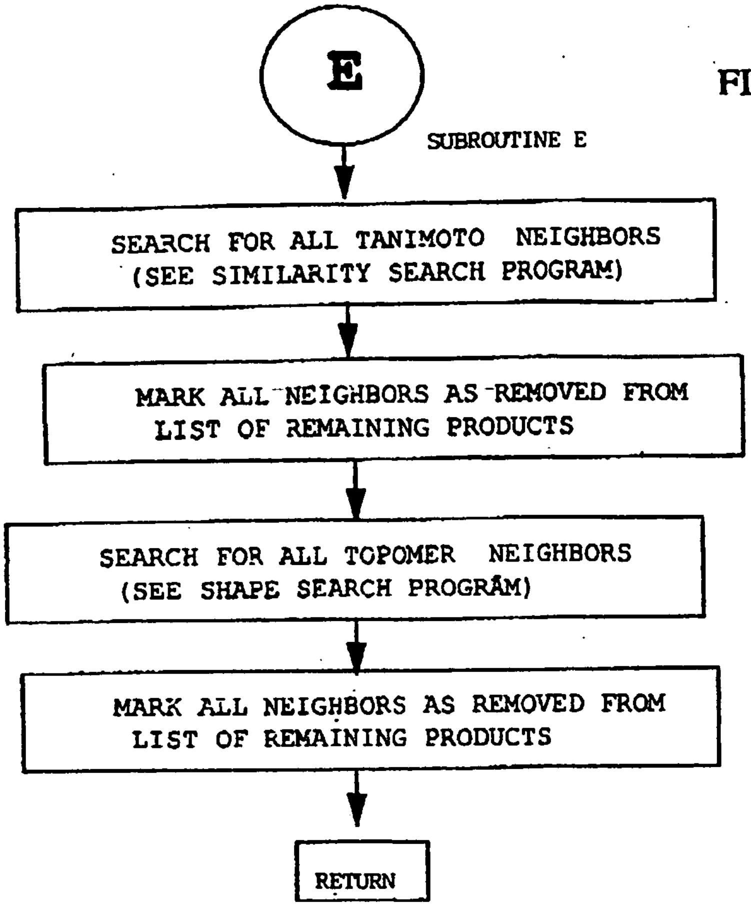

E
SUBROUTINE E
SEARCH FOR ALL TANIMOTO NEIGHBORS (SEE SIMILARITY SEARCH PROGRAM)
MARK ALL NEIGHBORS AS REMOVED FROM LIST OF REMAINING PRODUCTS
SEARCH FOR ALL TOPOMER NEIGHBORS (SEE SHAPE SEARCH PROGRAM)
MARK ALL NEIGHBORS AS REMOVED FROM LIST OF REMAINING PRODUCTS
RETURN

FIGURE 26

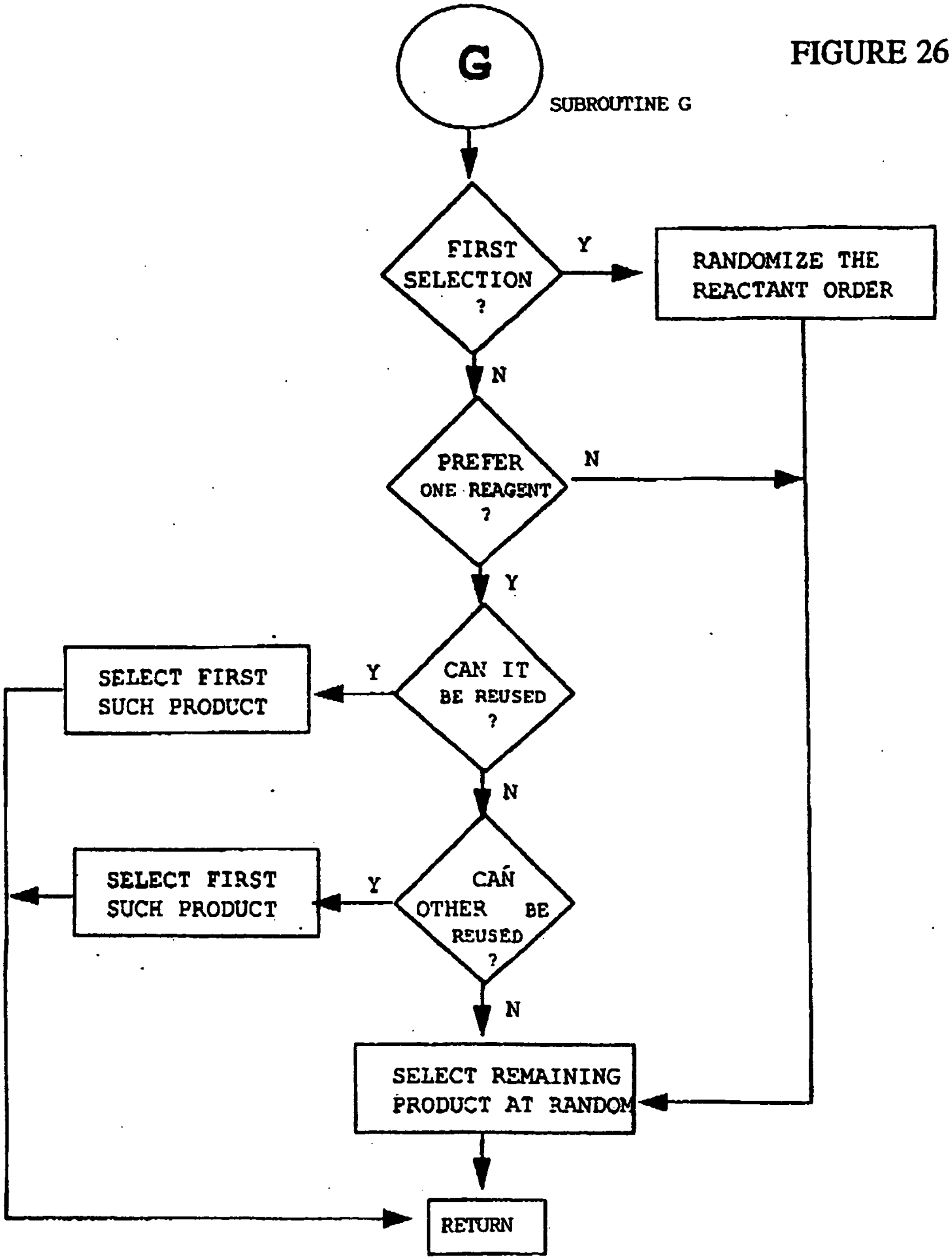

G
SUBROUTINE G
FIRST SELECTION ?
Y
RANDOMIZE THE REACTANT ORDER
N
PREFER ONE REAGENT ?
N
Y
CAN IT BE REUSED ?
Y
SELECT FIRST SUCH PRODUCT
N
CAN OTHER BE REUSED ?
Y
SELECT FIRST SUCH PRODUCT
N
SELECT REMAINING PRODUCT AT RANDOM
RETURN

FIGURE 27

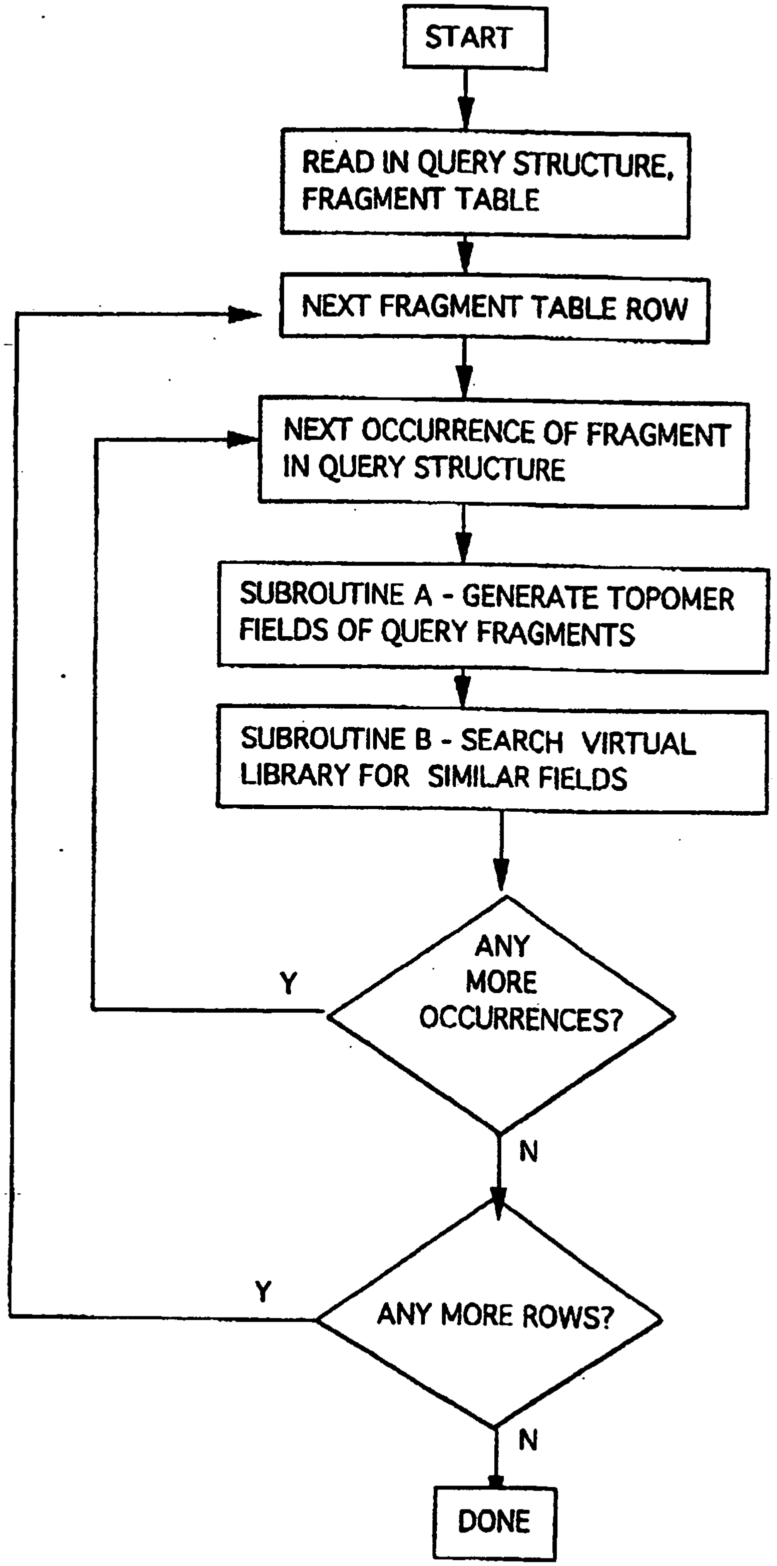
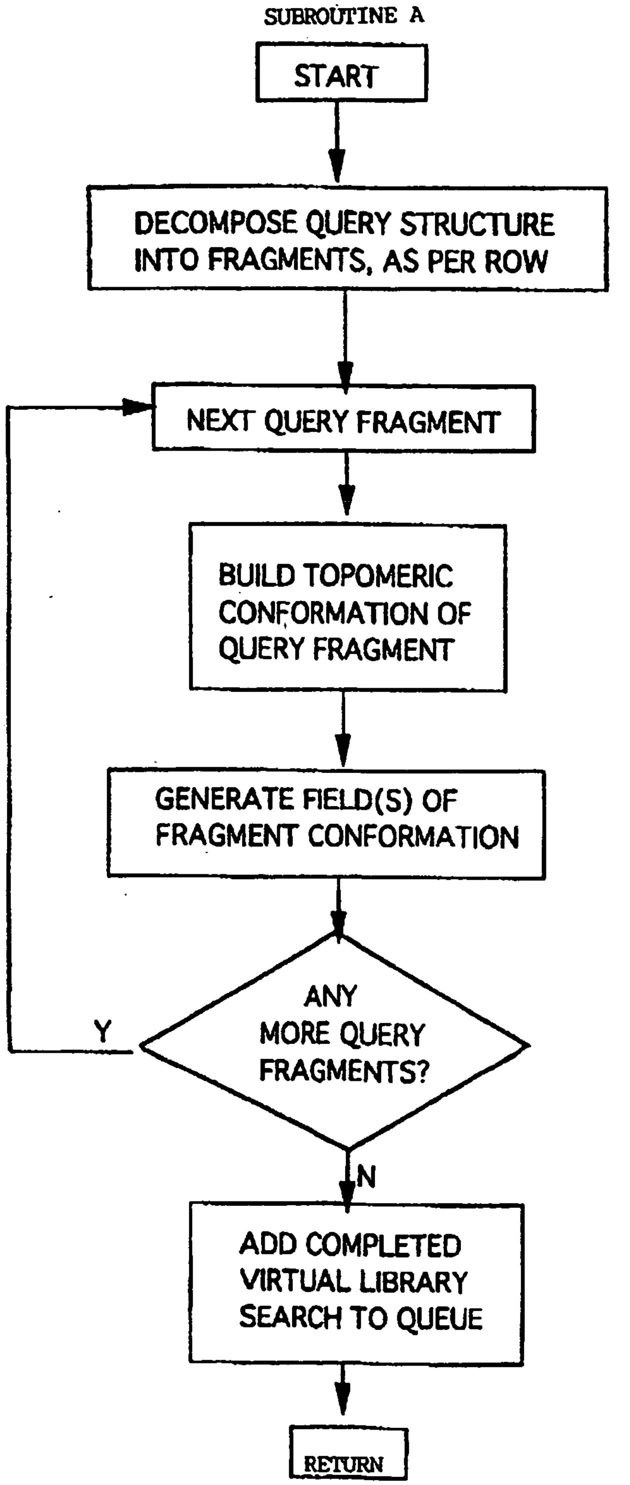

START
READ IN QUERY STRUCTURE, FRAGMENT TABLE
NEXT FRAGMENT TABLE ROW
NEXT OCCURRENCE OF FRAGMENT IN QUERY STRUCTURE
SUBROUTINE A - GENERATE TOPOMER FIELDS OF QUERY FRAGMENTS
SUBROUTINE B - SEARCH VIRTUAL LIBRARY FOR SIMILAR FIELDS
ANY MORE OCCURRENCES?
Y
N
ANY MORE ROWS?
Y
N
DONE

FIGURE 28


SUBROUTINE A
START
DECOMPOSE QUERY STRUCTURE INTO FRAGMENTS, AS PER ROW
NEXT QUERY FRAGMENT
BUILD TOPOMERIC CONFORMATION OF QUERY FRAGMENT
GENERATE FIELD(S) OF FRAGMENT CONFORMATION
ANY MORE QUERY FRAGMENTS?
Y
N
ADD COMPLETED VIRTUAL LIBRARY SEARCH TO QUEUE
RETURN

SUBROUTINE B

FIGURE 29

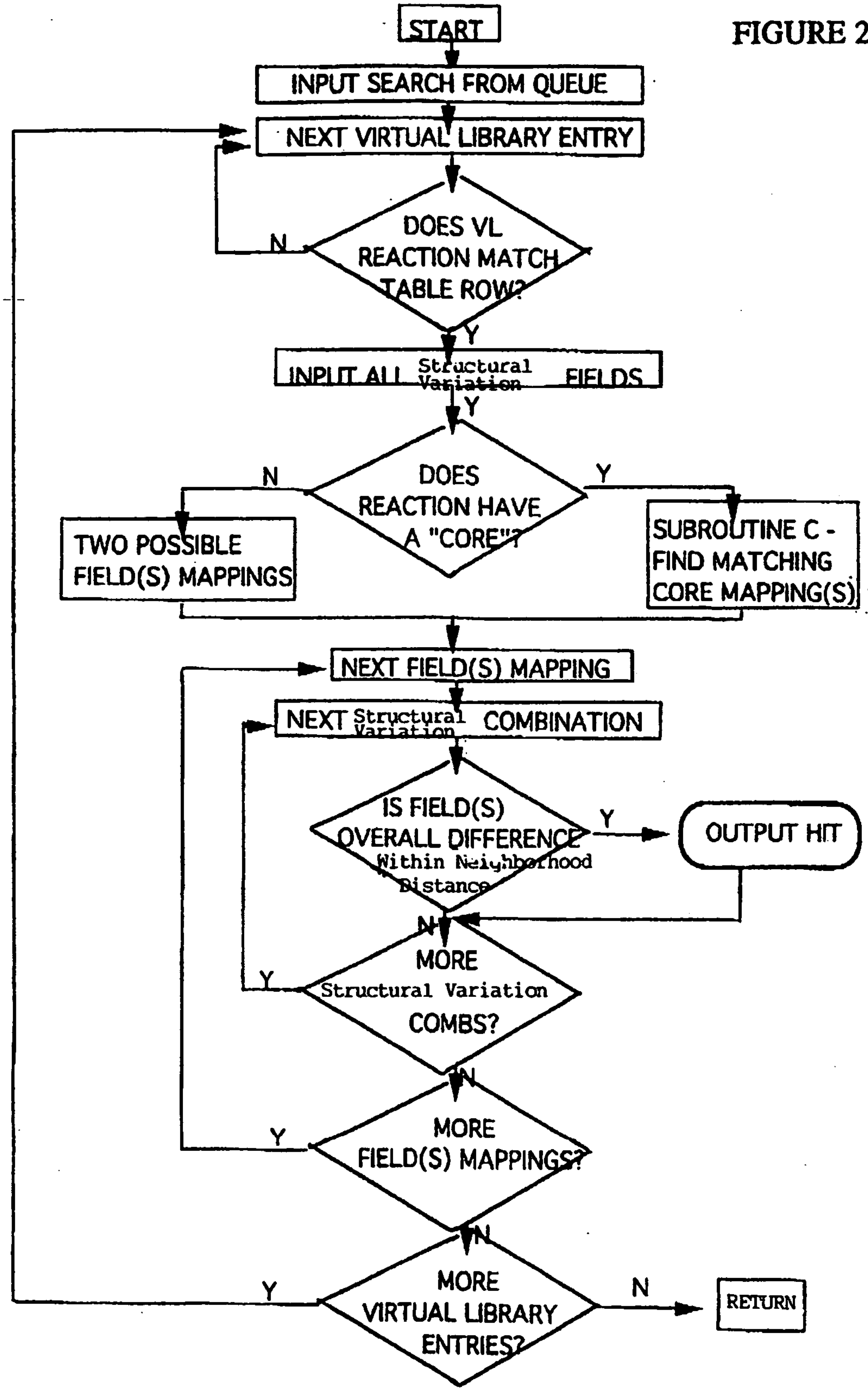

START
INPUT SEARCH FROM QUEUE
NEXT VIRTUAL LIBRARY ENTRY
DOES VL REACTION MATCH TABLE ROW?
N
Y
INPUT ALL Structural Variation FIELDS
Y
DOES REACTION HAVE A "CORE"?
N
Y
TWO POSSIBLE FIELD(S) MAPPINGS
SUBROUTINE C - FIND MATCHING CORE MAPPING(S)
NEXT FIELD(S) MAPPING
NEXT Structural Variation COMBINATION
IS FIELD(S) OVERALL DIFFERENCE Within Neighborhood Distance
Y
OUTPUT HIT
N
MORE Structural Variation COMBS?
Y
N
MORE FIELD(S) MAPPINGS?
Y
N
MORE VIRTUAL LIBRARY ENTRIES?
Y
N
RETURN

FIGURE 30

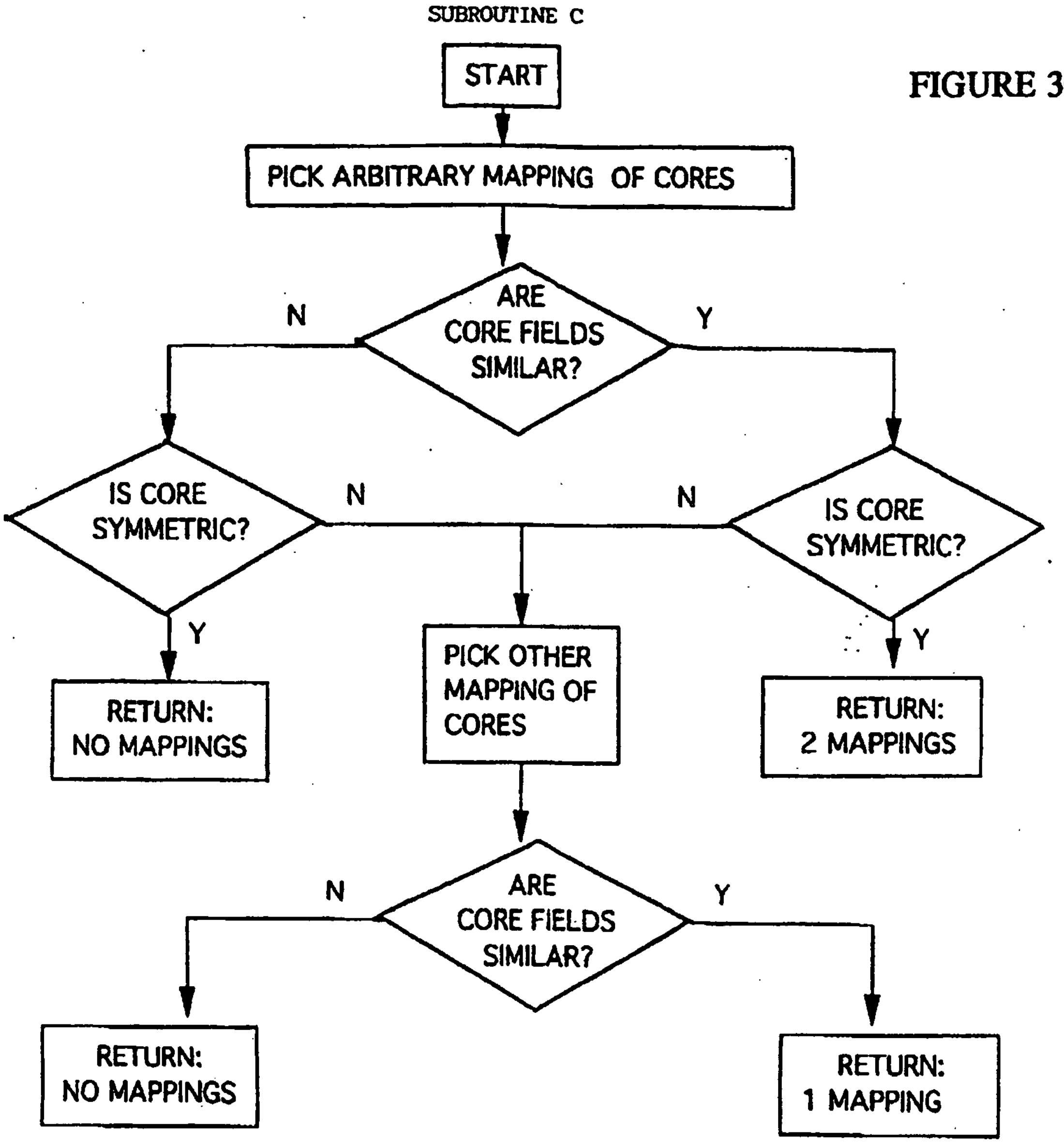
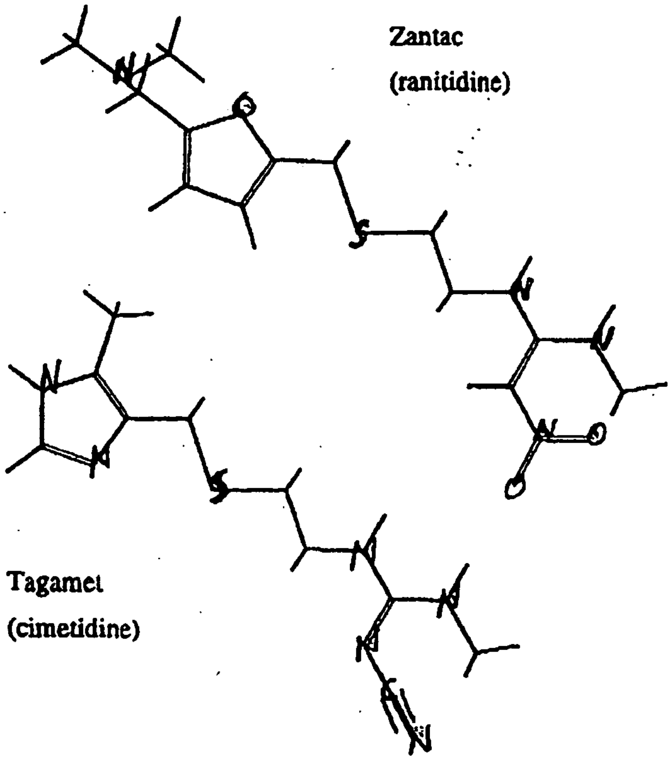

SUBROUTINE C
START
PICK ARBITRARY MAPPING OF CORES
ARE CORE FIELDS SIMILAR?
N
Y
IS CORE SYMMETRIC?
N
Y
RETURN: NO MAPPINGS
PICK OTHER MAPPING OF CORES
N
IS CORE SYMMETRIC?
Y
RETURN: 2 MAPPINGS
ARE CORE FIELDS SIMILAR?
N
Y
RETURN: NO MAPPINGS
RETURN: 1 MAPPING

FIGURE 31



Zantac
(ranitidine)

Tagamet
(cimetidine)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/01491

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 19/00
US CL :364/496
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 364/496-499; 395/601,616

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5,307,287 A (CRAMER, III ET AL) 26 April 1994, see abstract.	52-54
X	US 5,025,388 A (CRAMER, III ET AL) 18 June 1991, see abstract.	52-54
A	US 5,345,516 A (BOYER ET AL) 09 September 1994, see entire document.	1-94
A	US 5,270,170 A (SCHATZ ET AL) 14 December 1993, see entire document.	1-94

□ Further documents are listed in the continuation of Box C. □ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search

21 APRIL 1997

Date of mailing of the international search report

28 MAY 1997

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
EMANUEL T. VOELTZ
Telephone No. (703) 305-9714